

AD-A263 145



Cambridge Hydrodynamics Report 643

**DEVELOPMENT OF A NEW TECHNIQUE FOR
IMAGE RECONSTRUCTION, ENHANCEMENT,
AND VISUALIZATION**


— Final Report on AFOSR Contract F49620-90-C-0028 —

Steven A. Orszag, Principal Investigator

February 1993

**Cambridge Hydrodynamics, Inc.
P. O. Box 1403
Princeton, NJ 08542
(609) 683-1515**

**Best
Available
Copy**

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Development of a New Technique for Image Reconstruction, Enhancement, and Visualization (U)		5. TYPE OF REPORT & PERIOD COVERED FINAL 15APR90-14OCT92
7. AUTHOR(s) Steven A. Orszag, Principal Investigator		6. PERFORMING ORG. REPORT NUMBER CHI Report 643
9. PERFORMING ORGANIZATION NAME AND ADDRESS Cambridge Hydrodynamics, Inc. P. O. Box 1403 Princeton, NJ 08542		8. CONTRACT OR GRANT NUMBER(s) F49620-90-C-0028
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research 110 Duncan Ave, Suite B115 Bolling AFB, DC 20332-0001		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AFOSR-TR-93-0208
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) SAME AS 11		12. REPORT DATE February 1993
<div style="text-align: center;"> DTIC SELECTED APR 21 1993 S B </div>		13. NUMBER OF PAGES 150
		15. SECURITY CLASS. (of this report) Unclassified
16. DISTRIBUTION STATEMENT (of this Report) Unclassified — Unlimited APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) SAR		
18. SUPPLEMENTARY NOTES 93 4 20 101		93-08461 
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Triangulation Image reconstruction Image enhancement Edge enhancement Surface mapping Delaunay triangulation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The problem of describing and reconstructing a surface, both analytically and visually, has been extensively addressed in this work. Various algorithms have been developed, tested, and compared and several conclusions have been reached. They are: (1) if the surface information is given on a uniform grid, the surface is best described by a set of contour plotting routines that exploit the GL graphics libraries both in a solid as well as a wire-mesh framework; (2) if the surface information is given on a non-uniform grid, the surface can be described by Delaunay triangulation, a sophisticated graphics computer, or by interpolation back to a uniform grid. Numerous examples are given which show the advantages of these methods for specific applications.		

DD FORM 1 JAN 73 1473

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DEVELOPMENT OF A NEW TECHNIQUE FOR IMAGE RECONSTRUCTION, ENHANCEMENT, AND VISUALIZATION

Principal Investigator: Dr. Steven A. Orszag

1. INTRODUCTION

The problem of describing and reconstructing a surface, both analytically and visually, has been extensively addressed in this work. Various algorithms have been developed, tested, and compared and several conclusions have been reached. They are: (1) If the surface information is given on a uniform grid, the surface is best described by a set of contour plotting routines that exploit the GL graphics libraries both in a solid as well as a wire-mesh framework; (2) If the surface information is given on a non-uniform grid, the surface can be described by Delaunay triangulation, a sophisticated graphics computer, or by interpolation back to a uniform grid. Each of these methods possesses certain advantages over the others. In the event of insufficient information on the surface, two basic methods have been employed: (1) an algorithm to fill in "missing" information that employs Delaunay triangulation; and (2) interpolation amongst various grid points of known data assuming that the surface is relatively smooth. In the case of distinct missing information on outliers (regions of the surface possessing extremal properties missing from the original surface), no progress has been made.

2. TECHNICAL BACKGROUND

In general, a surface is described by a function, such as $f(x,y,z)=\text{const}$. Alternatively, a surface can be described by $z = g(x,y)$, with the understanding that $g(x,y)$ can take on multiple values of the surface folds back on itself. A typical area of interest to the Air Force is a surface reconstruction of photographs taken from high altitude. In some portions of the photographs where high resolution is needed, certain blurriness associated with absence of resolution may need to be improved. Furthermore,

r	
<input checked="" type="checkbox"/>	
<input type="checkbox"/>	
<input type="checkbox"/>	
n	
/	
y Codes	
and/or	
Dist	Special
A-1	

the need to remove some white noise from the picture may result in a distinct sharpening of the picture. Statistically, as long as the surfaces involved are very smooth, standard regression analysis could very successfully yield the desired results. However, the information that one wants to obtain are more often than not associated with some characteristics that are absent from the general smooth background of the images.

There are two distinct problems in surface reconstruction:

- (i) Decomposition of the surface into elements.
- (ii) Smoothly connecting these elements by interpolation.

The best solution to these problems is different for uniform grids than it is for non-uniform grids.

For non-uniform grids, the first problem can be solved by the Voronoi-Delaunay construction. Each vertex is surrounded by a Voronoi cell, defined as that cell that contains points which are closer to this vertex than to any other. The boundaries between neighboring Voronoi cells are lines, equidistant from a pair of vertices. Connecting all such pairs by links generates a Delaunay triangulation. We exploit this geometrical construction by observing that each triangle (abc) involves vertices which are Voronoi neighbors. There is a point Y where all three Voronoi cells (a,b,c) touch. By construction, all three vertices a,b,c are equidistant from Y so that these points are on the circle centered on Y. No other vertex can exist inside this circle by definition of the Voronoi cells. This property has been used in the construction of an incremental algorithm for Delaunay triangulation. We have developed an algorithm that accomplishes this triangulation recursively. Strictly speaking, the task is to add a point to an existing triangulation of N points. An added point exists and is contained inside one of the existing triangles. If we connect the new point to the vertices of the triangle in which it lies, we get the zeroth approximation of the triangulation of N+1 points. This triangulation is good away from the new point, but must be rebuilt in the local neighborhood of the new point. The maximum area of reconstruction is given by the

circle centered on Y, and in general, it requires only a finite number of bond flips (triangle reconstructions). A recursive flipping algorithm can be derived and is very fast. The most costly operation in this incremental Delaunay triangulation is the search for the triangle surrounding the new point. Straightforward scanning takes $O(N)$ time, but preprocessing the data accelerates the search by the following strategy.

One can order the vertices along any specific curve so that neighbors on the curve will be always close to each other on the plane. (This is a version of the traveling salesmen problem.) Such an ordering of initial points takes $O(N \log N)$ time. This ordering is needed to be performed only once, after which various progressive surfaces can be drawn using the same coordinate system. Degenerate cases, when the points are too close to each other or the $(N+1)st$ point is too close to the side of the Nth triangulation, are treated separately to prevent numerical instabilities.

The second approach is based on the graphics implemented in the graphics terminals of Evans and Sutherland. The points are loaded on the system and are displayed simultaneously with x,y, z coordinates all belonging to the same surface. The surface can be rotated around each of the three axes and cutting planes have been introduced to isolate specific projections of the surface on a plane of choice. On a plane of choice, curves have been interpolated using optimal bicubic splines. The points are added on the splines and the files of the individualized planes get re-assembled. A perpendicular plane to the original set of parallel planes is being introduced and the added points in each of the previous planes is compared to the initial set of points on this cut plane. A least-squares estimator is performed to decide on admissibility or inadmissibility of the added points. If the set of points added to the global grid is admissible, a point-by-point elimination process for the added points is invoked until the set of added points becomes admissible. The set of points that are then admissible per

perpendicular plane are then added to the global grid and the process may be repeated until no more points are admissible. At that stage, the entire file is re-loaded on the graphics system and a surface reappears. The surface can be described by the actual collection of dots, wire-mesh, or by standard graphics shading routines on the graphics processors. We are now in the process of adapting these algorithms to the SGI GL libraries.

The third case for the non-uniform grid can be handled in two different ways. First, a uniform grid is imposed in the x-y plane. At each grid point, a characteristic radius is introduced. Each point in the x-y plane, x_i, y_i is taken into account. The number of points p in the enclosed domain is recorded and a p th Lagrange interpolation scheme is introduced to compute the value z associate with the said grid point. When all the uniform grid point values have been obtained, the interpolation scheme is checked against each existing original point. In other words, each original point is attempted to be recomputed to check for accuracy. This is done with variable orders in order for outliers estimates to be rejected. In the second case, at any vertex in the Delaunay triangulation, an arbitrary number of triangles can meet and the property of smoothness at the vertex requires a large number of local parameters. We have the following construction that satisfies all the required smoothness conditions. The Gauss formula for planes of the corresponding triangles meeting at a vertex has been used. The smooth function so obtained passes through all its neighbors. This function is usually a sum of about six terms. Three of such functions are connected inside each triangle by a similar Gauss interpolation resulting in an explicit local set of rational functions within each triangle matching their neighbors with any prescribed smoothness. The addition of the interpolated points to the original triangulated surface is composed by the set of points at the center of the Delaunay circles, i.e. vertices of the Voronoi graph which are always located in the voids of the Delaunay triangulation. Adding these circle centers does not

require a search and is numerically stable. It roughly doubles the number of points omitting the points whose Delaunay circle radius is smaller than a given criterion to avoid condensation of points. After a few steps, the plane is completely covered and the process is complete.

The issue of describing a surface on a uniform grid is different in nature. An aerial photography or topographic description of a surface can in principle be described on a uniform grid since the data gathering can be digitized according to the data gatherer. At this juncture, a missing observation on the uniform grid is interpolated using standard Lagrange interpolations and data contained within noise can be filtered out by invoking standard FFT techniques. In particular, a combination of low and high-pass filters can isolate specific structures by sharpening the domain of resolution. It is believed that this technique may have the potential of image enhancement.

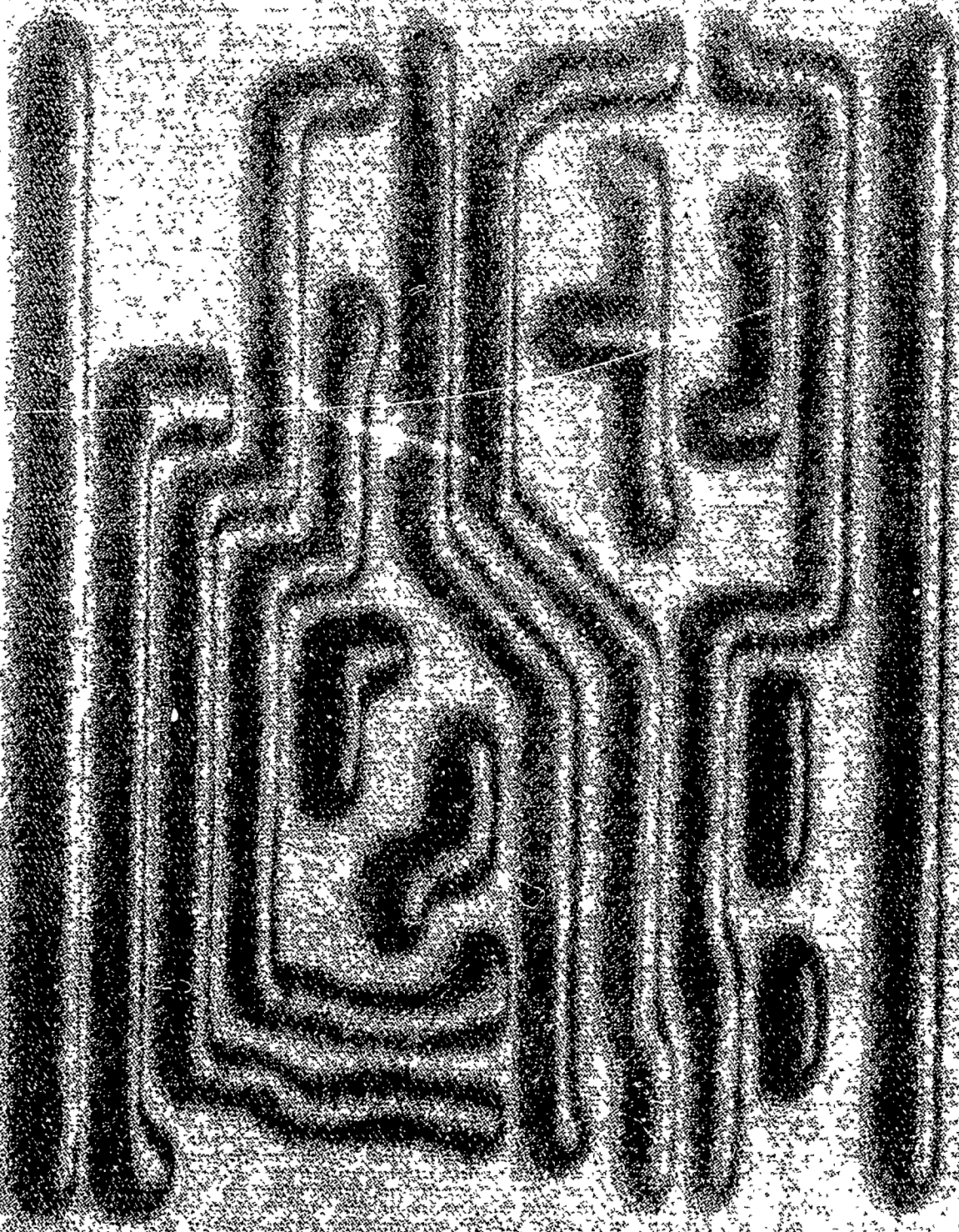
One of the major goals in this study is the ability to characterize an irregular surface, in particular, non-uniform surfaces obtained either from exterior sets of observations or from solutions of partial differential equations of moving fronts or equipotential surfaces. For example, the system of detecting irregular conductivity of an internal human tissue is dependent on surface potential and conductivity. A solution of the potential problem inside the bulk with isolation of equipotential surfaces is hoped to improve visualization of internal irregular tissues. It is for such surfaces that the non-uniform grid methodologies developed here will be useful.

3. IMPLEMENTATION

We have developed the algorithms specified in Tasks 1-3 of the original proposal. These are exemplified by the attached source code in Appendix I. For Task 4, we have developed a menu-driven user-based interface that is displayed with all its functions in

the attached plots. The use of this interface and code is elementary and requires minimal learning time. The code interface is implemented by use of a mouse within the graphics window. For Task 5, we provide various sets of examples of non-uniform as well as uniform grid systems. The uniform grid graphics visualization displays a system borrowed from microelectronics of contact hole printing when the surface of the electromagnetic energy intensity is displayed above a given surface. Multi-resolution is used in these calculations to address the requirements of circuit designers. Furthermore, we attach two different circuit designs (see Figs. 1-2) obtained for uniform grid implementation of two separate logic patterns for wiring and memory triggers. In both cases, GEOFORM gave a reasonably satisfactory description of the circuit and its associated resolution.

For the non-uniform grid systems, we have tested this method on geologic data, Martian topography, vortex sheets in hydrodynamics and also on a sculpture reconstructed from a few hundred points. Sometimes smoothing was needed, i.e., the points were iteratively moved towards the local second-order surfaces, least-square fitted to the nearest neighbors. Since this fit was local, it did not erase the small details at the surface of larger structures. Everything was smoothed at its own scale. This algorithm is surprisingly efficient. The Mars topography reconstruction required only 2% of randomly chosen data points. The sculpture was reconstructed from just 600 points, and the roll-up of a vortex sheet was simulated using only 1000 points. In Figure 3, we display the Delaunay triangulation of the initial mask. The reconstructed surface is displayed next to this triangulation and the internal variations in the density of the triangulation manifests in different shading to create the necessary visualization of the actual original mask. The other two segments of Figure 3 involve the reconstruction of very sharp mountain peaks on the Asian continent where the triangulation density indicated initially a sharp gradient that led to the reconstruction of the Himalayan peaks.



**Figure 1: GEOFORM plot of aerial
image of 365 nm microlithography**

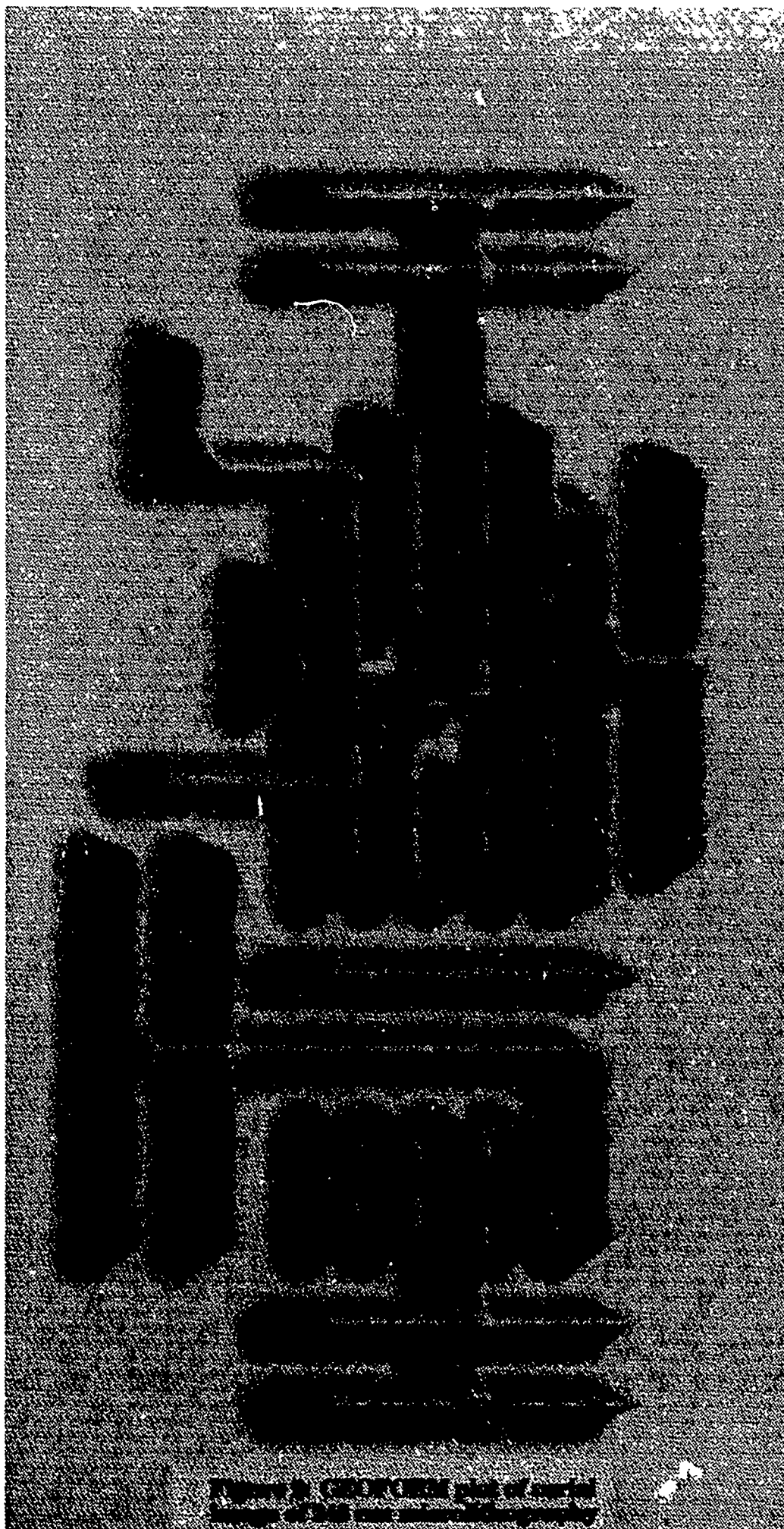


Figure 2. CINEFORM plot of aerial
images of 248 east-southwest-south

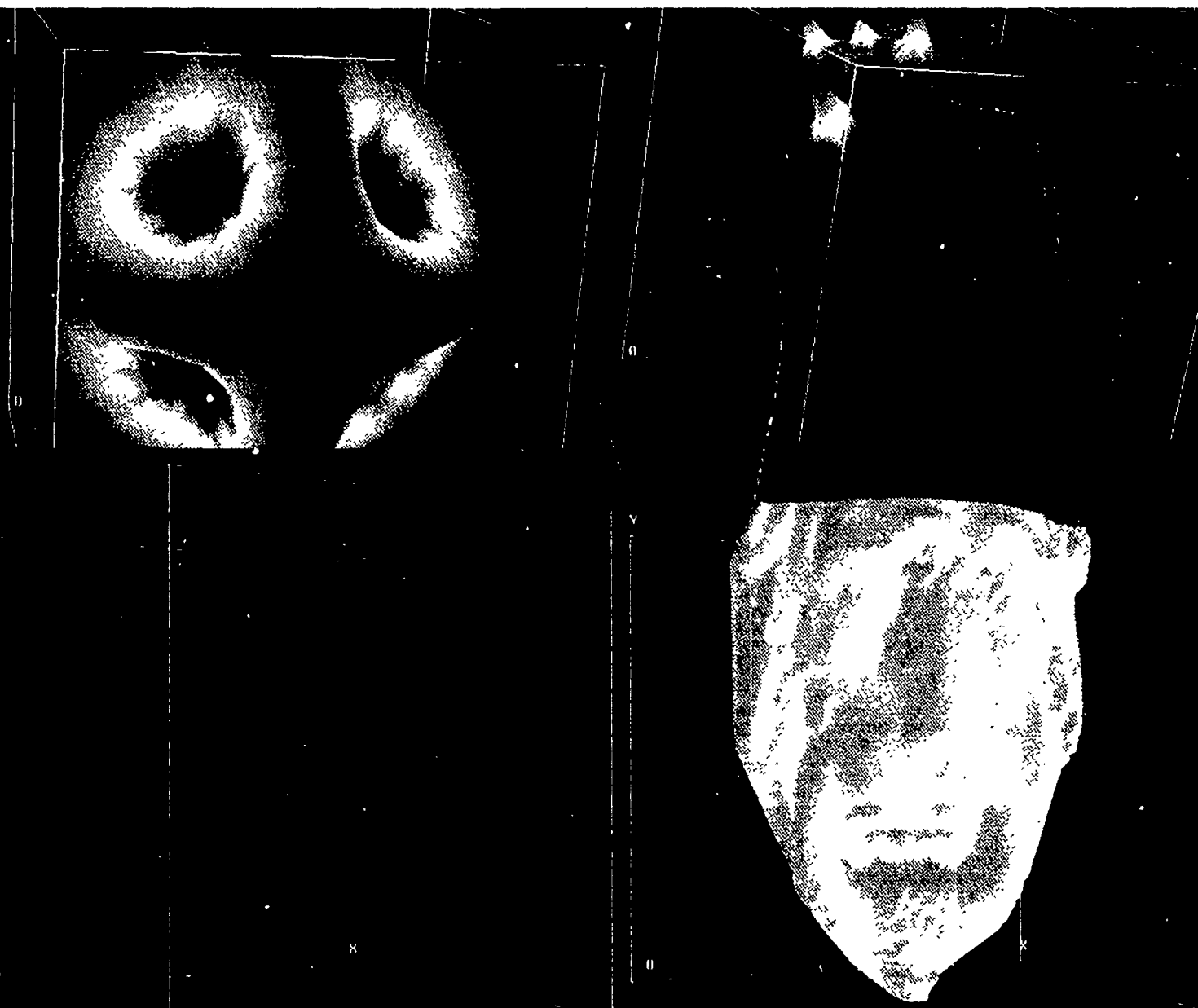


Figure 3: Examples of GEOFORM applications

Higher intensities (higher heights of the mountain peaks) are illustrated in red to emphasize the reconstructed of the missing peak. This technique could be very useful to reconstruct surfaces obtained from high altitude with blurred resolution at the extremes. The fourth quadrant of Figure 3 illustrates a reconstructed sphere with cavities inside. It could be particularly important to use this technique in the simulations of porous media where specific flow channels can be obtained to determine porosity of new materials. The dynamics of pore sizes is under current investigation.

In Figures 4-5, we display graphically the detailed mechanism of edge detection in the GEOFORM code. It is menu controlled and specific points associated with the edge of a given surface are determined locally or globally. In particular, a zoom window provides the ability to zoom in on an edge of interest. The particular example given is that of refining the edge of a 'rabbit' shaped region. It is apparent that even rough resolution suffices to solve the problem. A second graphical option associated with a collection of points is the imposition of the interior Delaunay triangulation embedded in these points.

To explain how we do this, let us consider a collection of points which lie along lines. The points may be given in any order. Our objective is to divide the set of points into subsets, such that all the points in one subset lie along a line. In other words we want to reduce our initial set of points to a set of clusters of close points. We then compress these clusters to "skeletons". These skeletons are represented as graphs with logical links between points. After this compression, one can either compare graphs with a lookup table of standard elements or simply transmit the graph. At each link of the graph one can store the number and mean dispersion of the compressed nearest points, so that one can return to the thick lines after transmission by randomly distributing points along the links. As compared to standard methods, this one seems to be especially well suited for graphic objects such as handwriting, fingerprints, space photos or maps.

POINTS

TRIANG

SHOW ALL

SHOW 410

QUIT

NORMAL

ZOOM

WINDOW

Figure 4: Edge detection: rough
boundary of 'rabbit'

POINTS

TRIANG

DEMO

DEMO M

D MAX

D MIN

D_M_T

D_M_T_Q

QUIT

ALL: 380; DELETED: 323; REMAIN: 57; COMPRESSION 6.666667 times

NORMAL

200%

WINDOW

Figure 5: Edge detection: refined
boundary of rabbit using GEOFORM

We use the following method. First, we construct the Delaunay triangulation of the set of given points, and then we begin a transformation of the triangulation which will extract the points lying along lines. We have developed a set of functions described below that we apply to our triangulation. The function DELETE MAX EDGE deletes the longest edge of the Delaunay triangulation. The function DELETE MAX EDGE IN TRIANGLE deletes the longest edge, which does not "disconnect" the triangulation, i.e. we delete the edge only if each of its end points has at least three neighbors. The function does nothing if such an edge does not exist. The function DELETE MIN EDGE deletes the shortest edge of the Delaunay triangulation. The function MERGE MIN EDGE merges the end points of the shortest edge into one point. This new point is placed at the center of mass of the two old points taking into account the number of points which were merged into the points before. The function MERGE MIN TRIANGLE merges the three vertices of the smallest triangle (by area) into one point. The new point is placed at the center of mass of the three vertices, taking into account the number of points which were merged into the vertices before. With these functions in hand, it remains to determine the order of applying them to the triangulation. To explore this question, we developed a tool implemented on IRIS workstations which allows us to choose which function to apply to the triangulation at any time to achieve the best line extraction. The tool collects the information about all steps and prints out the statistics on the working process. This helps to determine the best strategy for applying our functions.

The reconstruction of Martian surfaces is displayed in Figures 6-11. These illustrate extremes obtained from higher Delaunay triangulation densities with missing peaks. A solid surface with proper shading is reconstructed to visualize the actual surface obtained from discrete pieces of information.

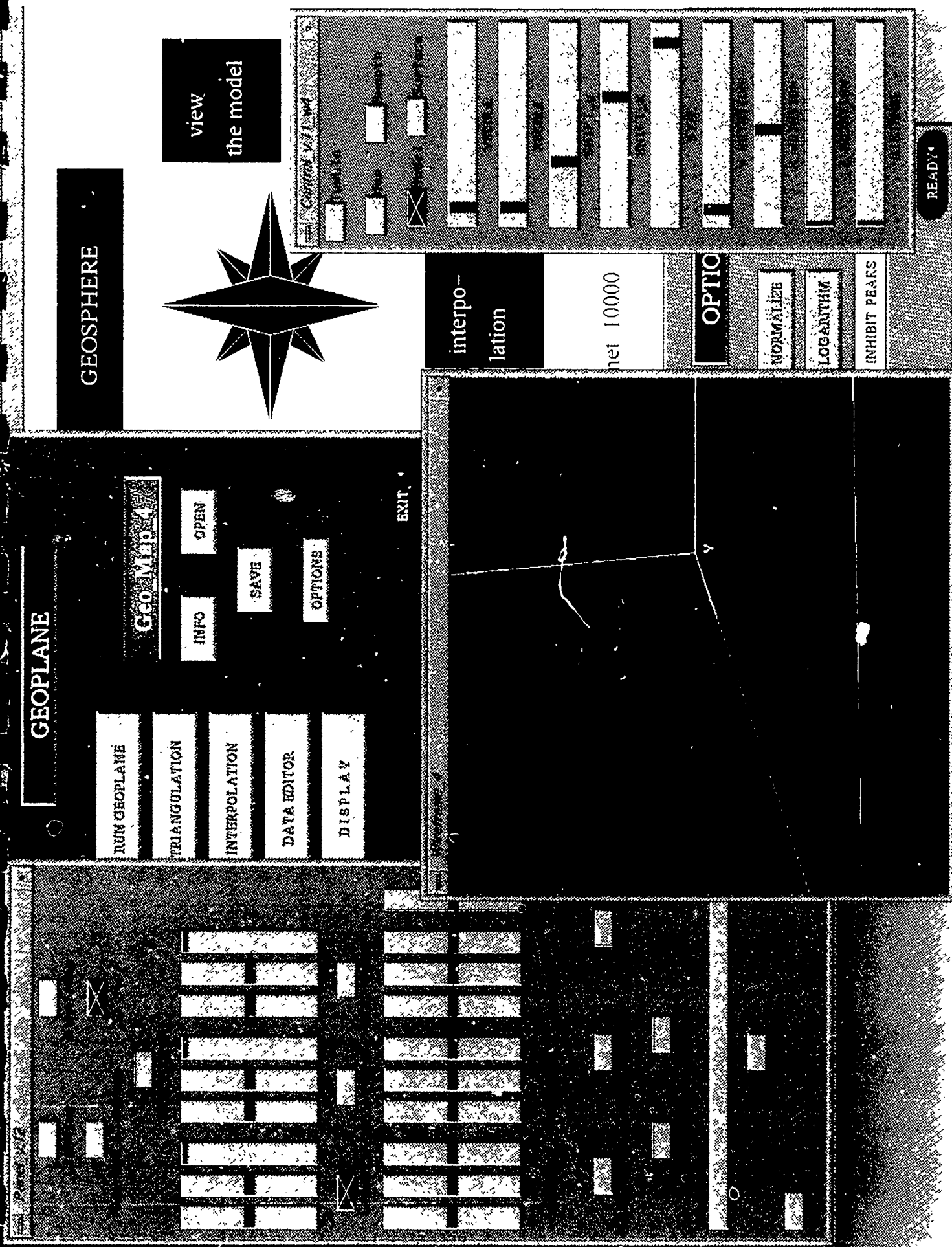


Figure 6: Menu-driven wire-mesh triangulation and refinement

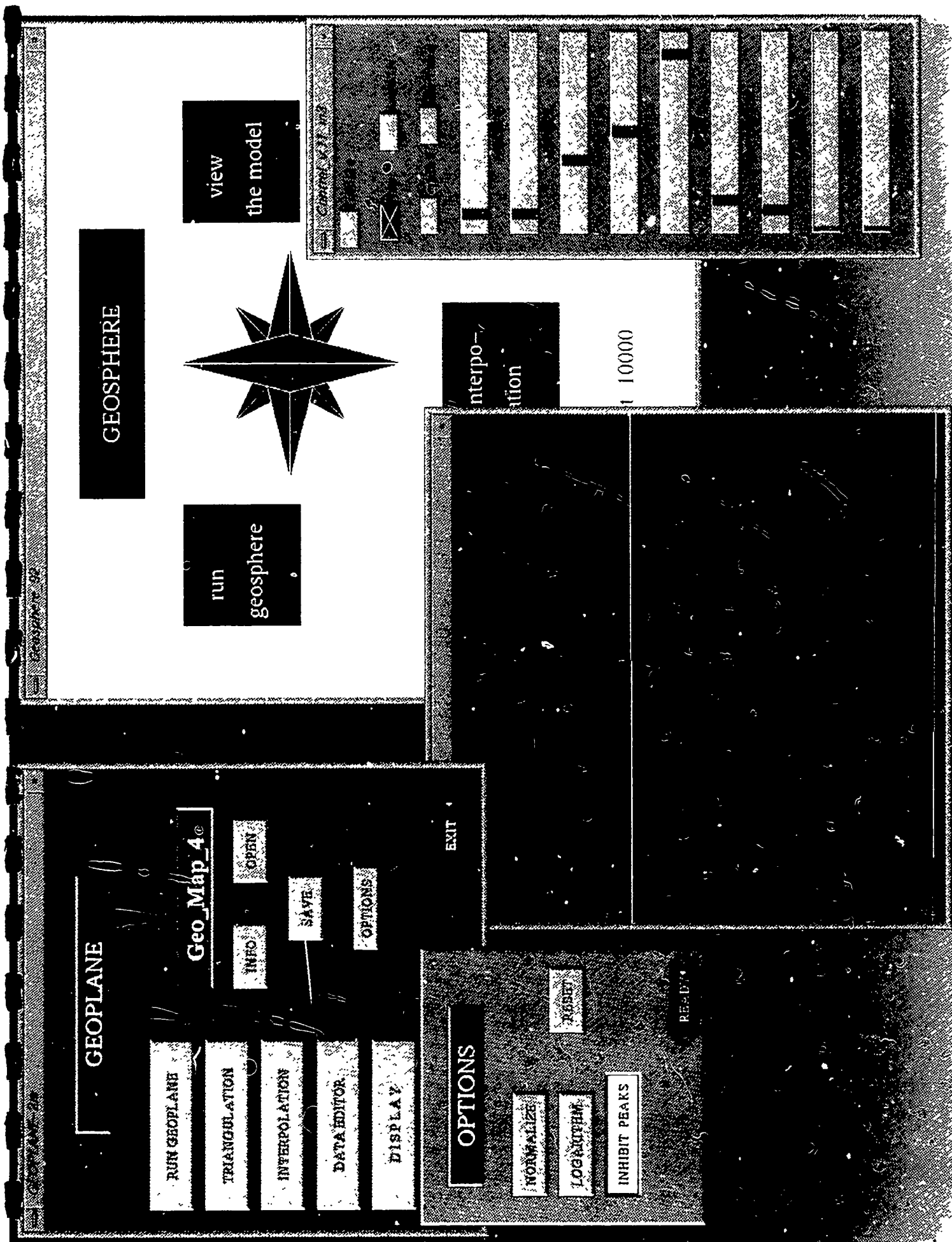


Figure 7: Menu-based control environment of GEOFORM

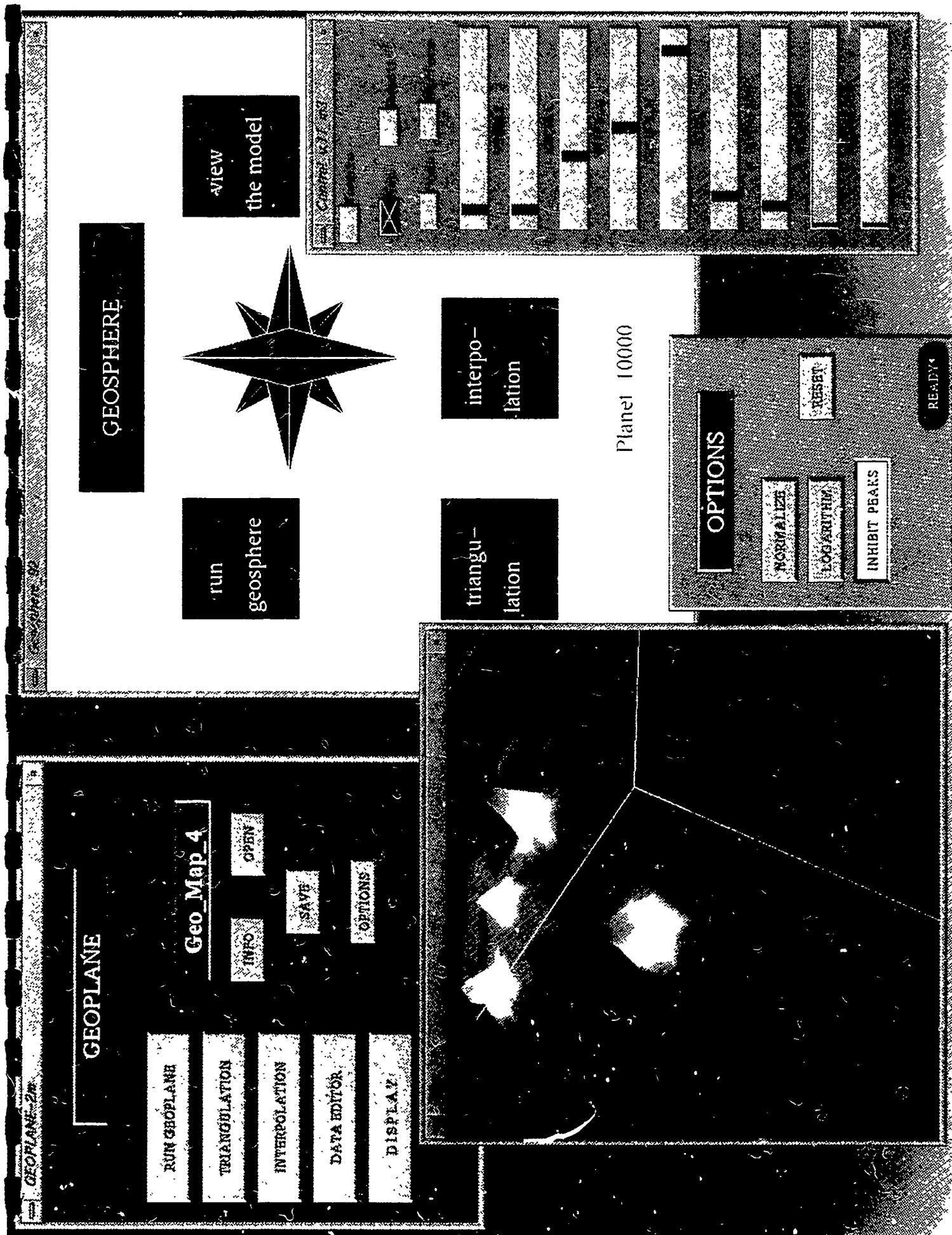


Figure 8: Menu-based topo-mapping environment of GEOFORM

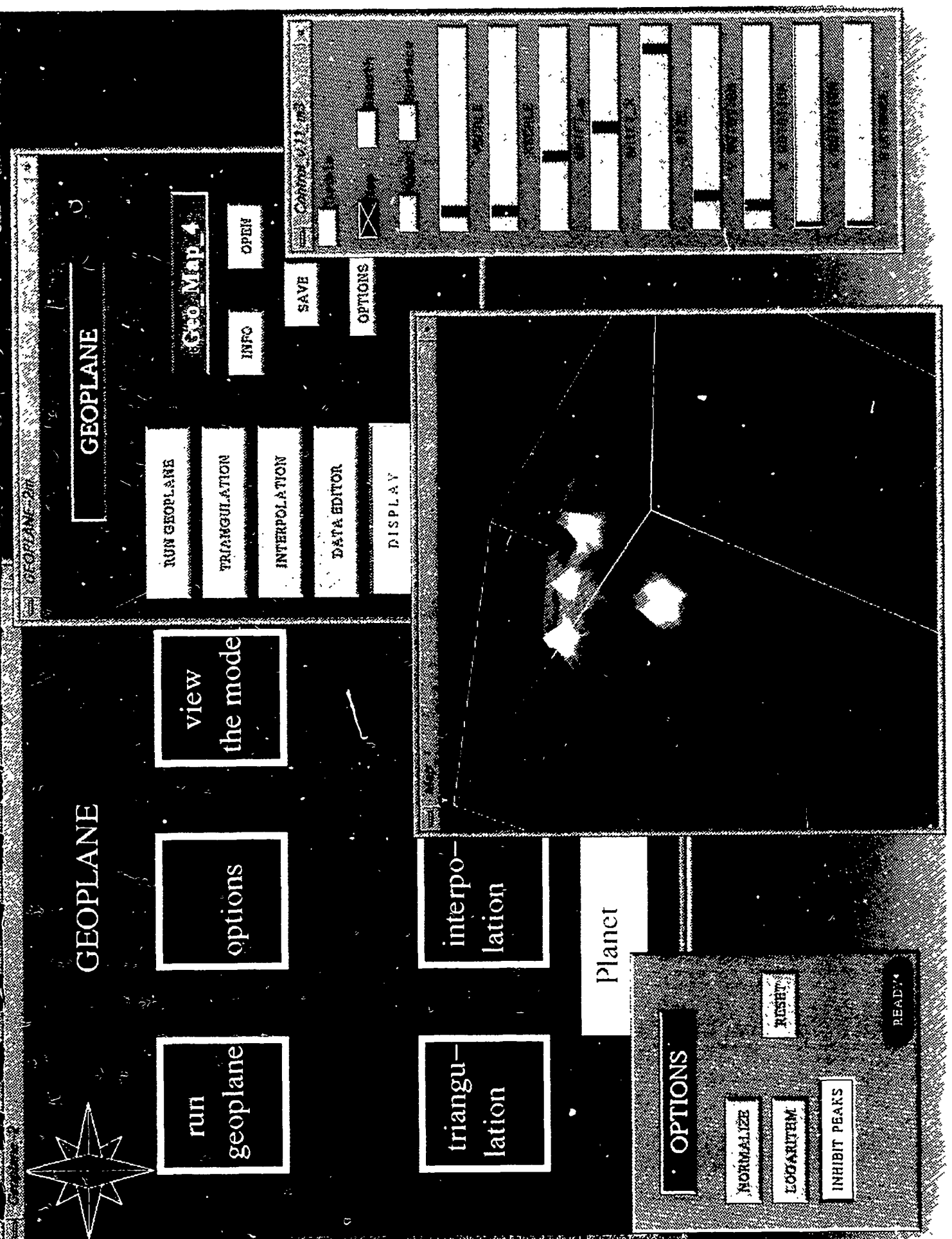


Figure 9: Menu-based mapping environment of GEOFORM

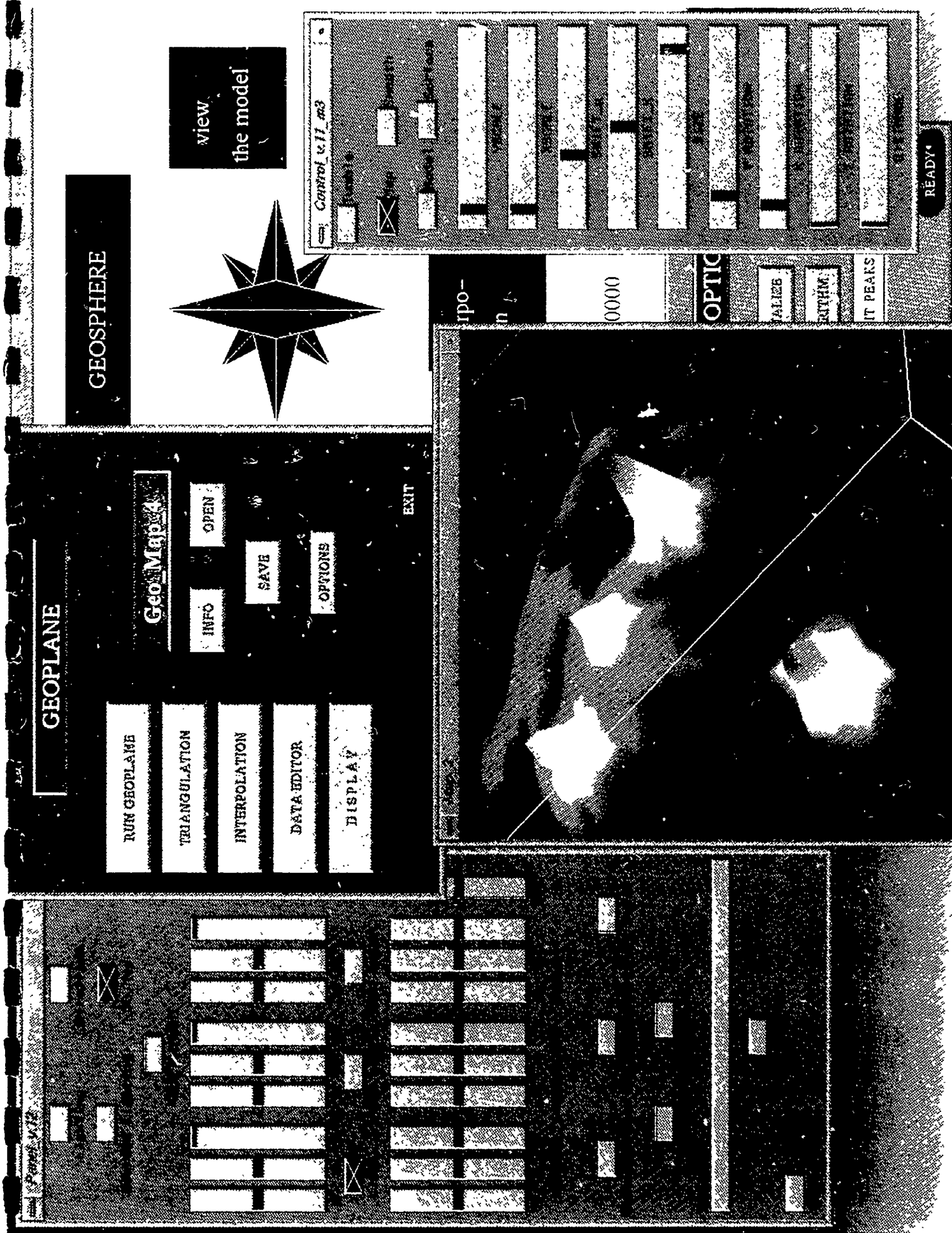
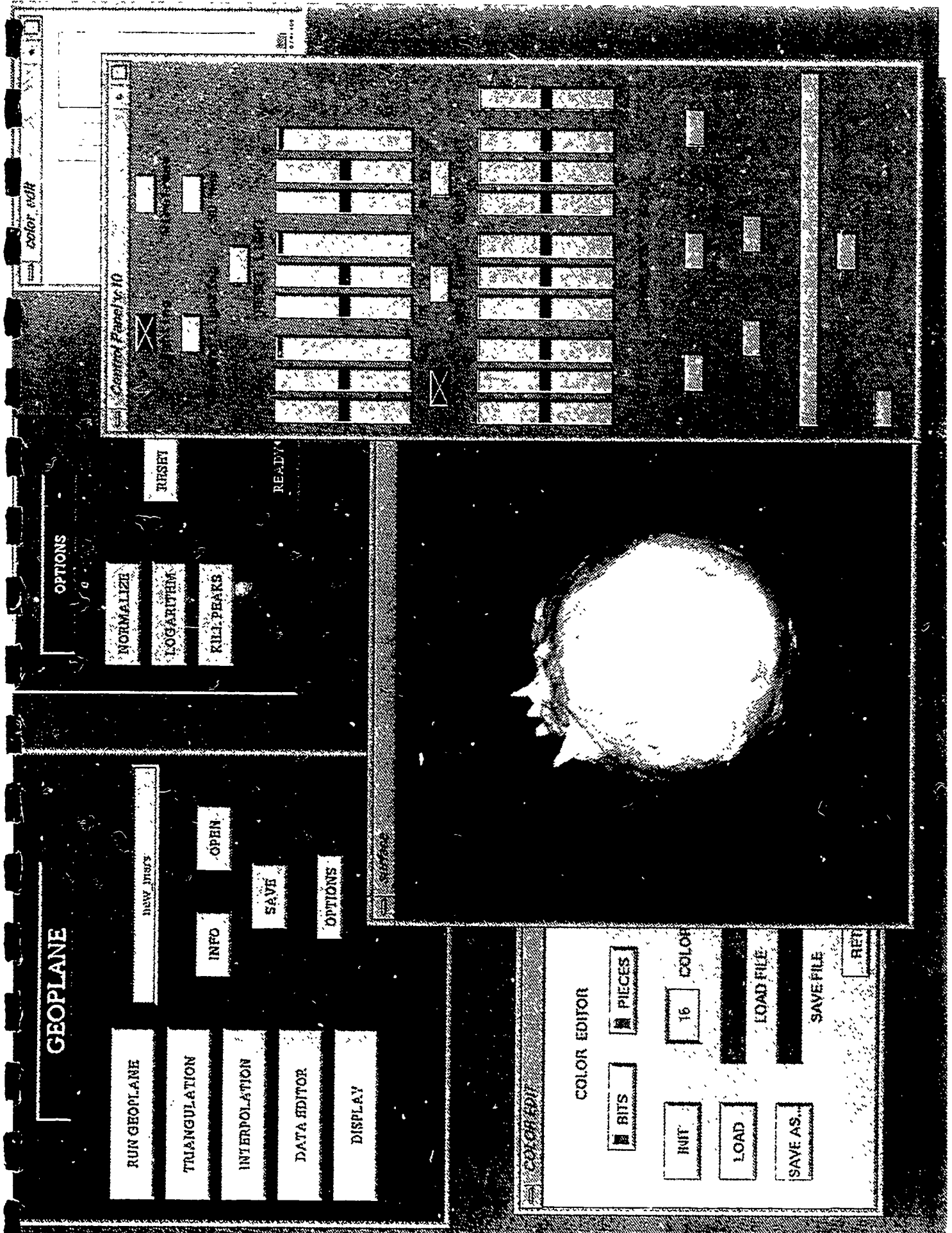


Figure 10: Menu-based zoom-in environment of GEOFORM

Figure 11: Menu-based lighting environment of GEOFORM



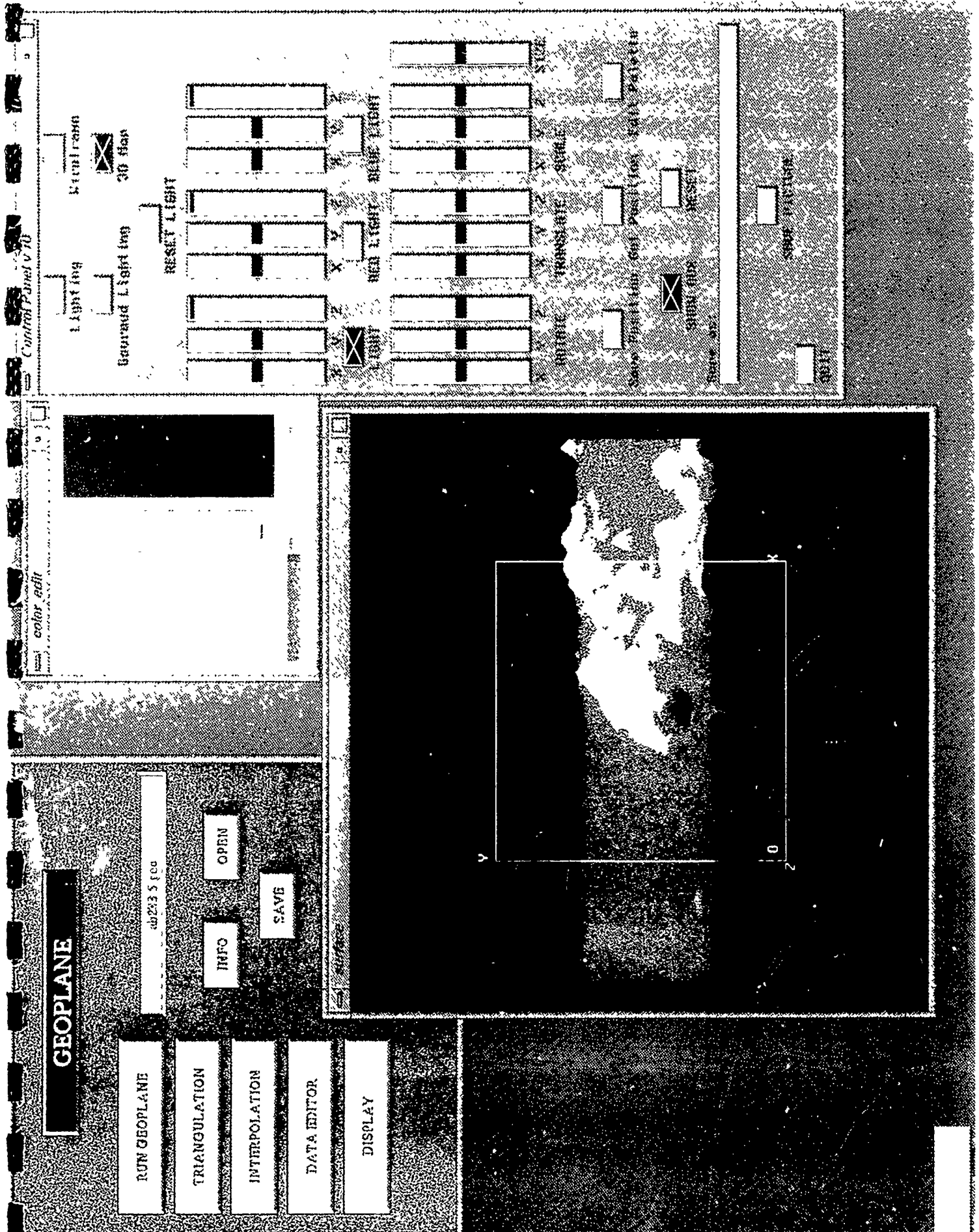
Last, but not least, we describe some of the features associated with the menu-driven graphics system of GEOFORM. In the GEOPLANE panel, we have a button in which the file name to be loaded is inserted. Then, triangulation takes place upon clicking on the triangulation button. After the determination of missing points takes place, the interpolation button gets clicked. Then the data editor is triggered and the display provides the surface. The button labeled 'option' allows either normalization, smoothing by killing peaks, or a logarithmic scale contraction.

Another panel, the control panel, allows changes of light intensities, changes of colors, changes of individual rotation angles, and allows saving of files under the button 'Save Picture'. The color editor selects the bit of the graphics involved and allows the insertion of shading under the control panel instructions. These panels, as well as their sub-panels provide complete graphics control of the surface to be displayed either as a wire mesh or as a shaded solid surface.

The remarkable flexibility of displaying results is best illustrated in the hydrodynamic pictures provided below. In Figures 12-15, we apply GEOFORM to the visualization of flame fronts in turbulent flows. In Figures 16-20, we apply GEOFORM to the visualization of random of multi-frequency Rayleigh-Taylor instabilities occurring at density interfaces.

In conclusion, we have developed new algorithms for data preprocessing and organization, triangulation and interpolation. Numerous tests verify the efficiency and numerical stability of these new algorithms. In particular, the surfaces are glued together from small elements, depending on a finite number of neighboring points, but smoothly connected to each other. A listing of the GEOFORM family of codes is given in the Appendix.

Figure 12: Propagation of flame front in turbulent channel flow



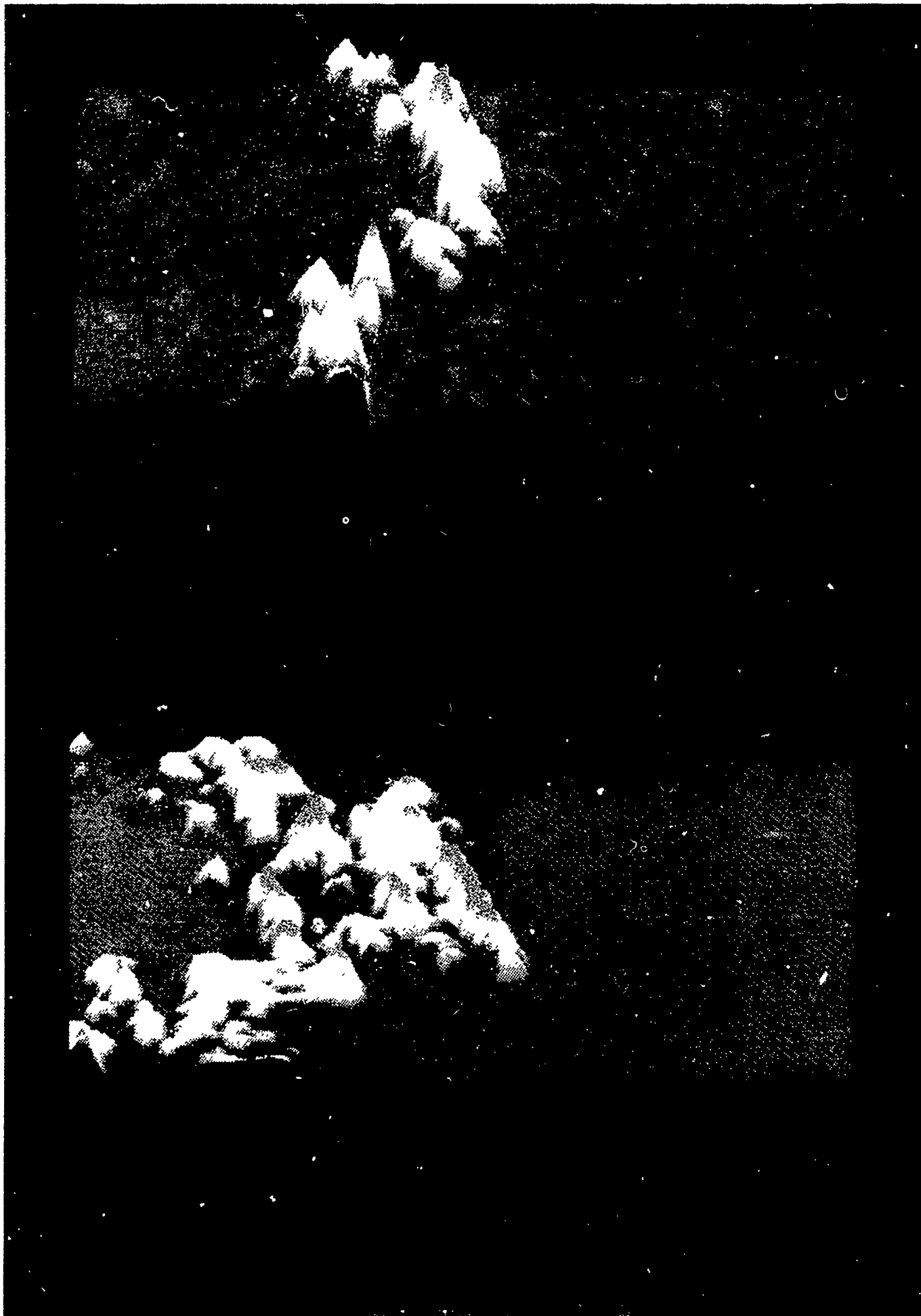
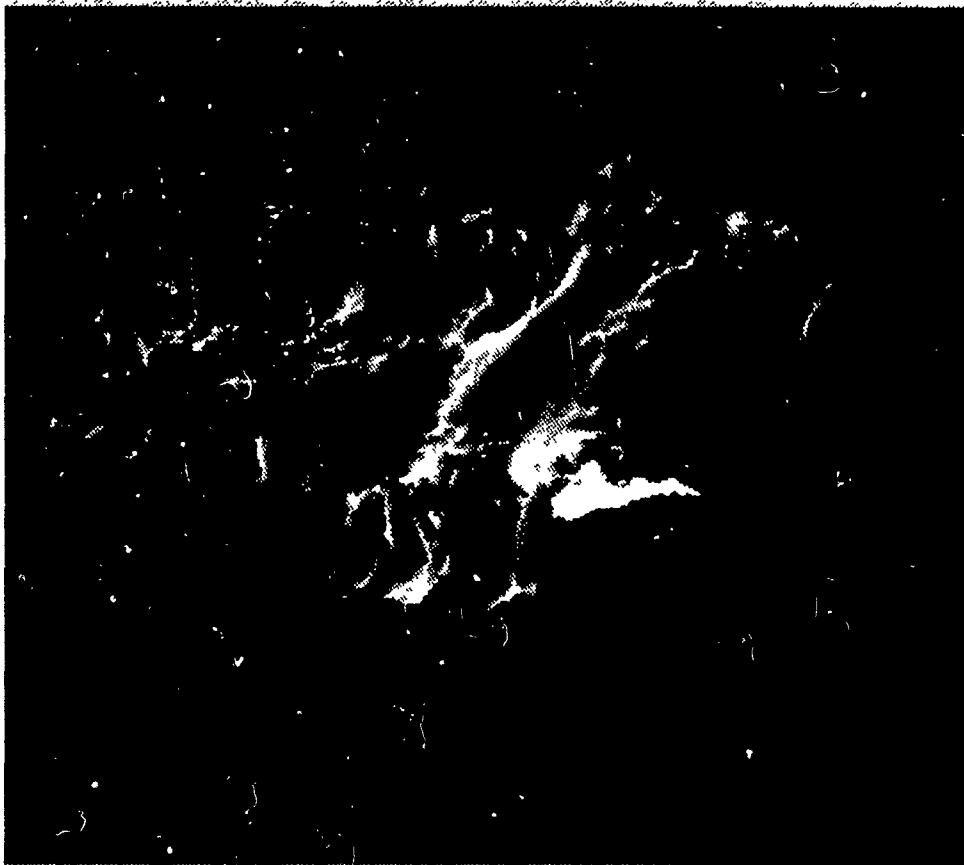
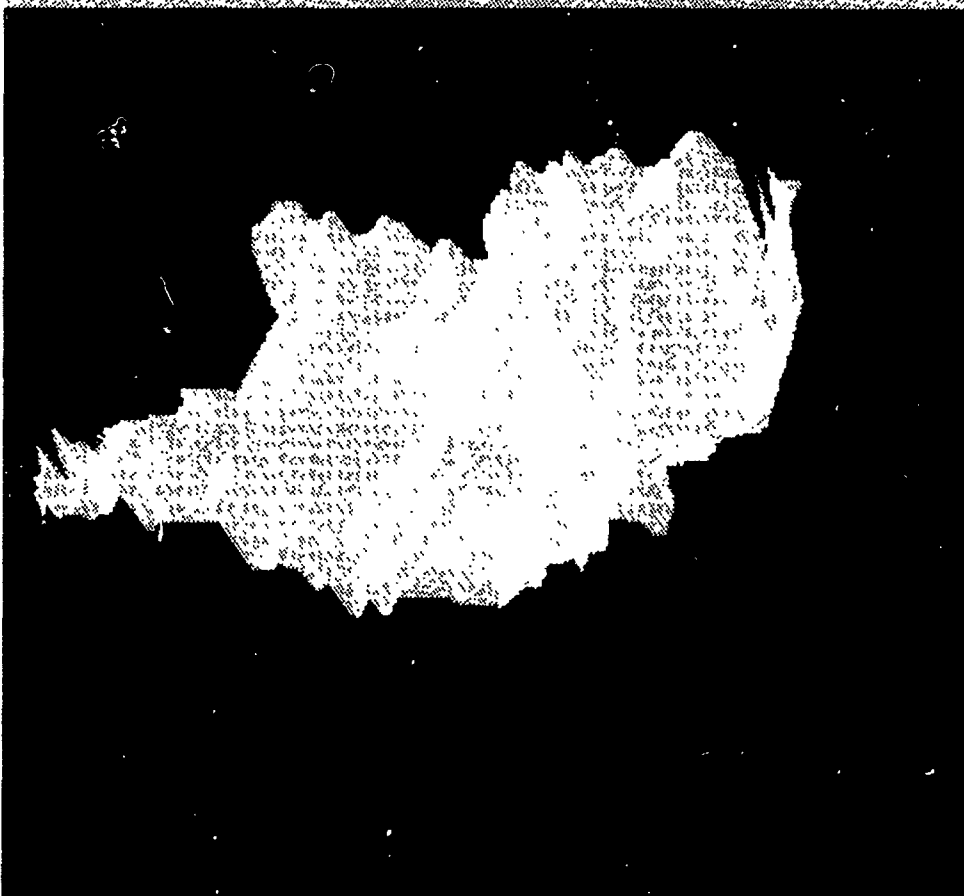


Figure 13: Propagation of flame
front (late time)



Flame Front



Maximum Temperature Front

Figure 14: GEOFORM view of flame
obtained using RNG-LES simulation

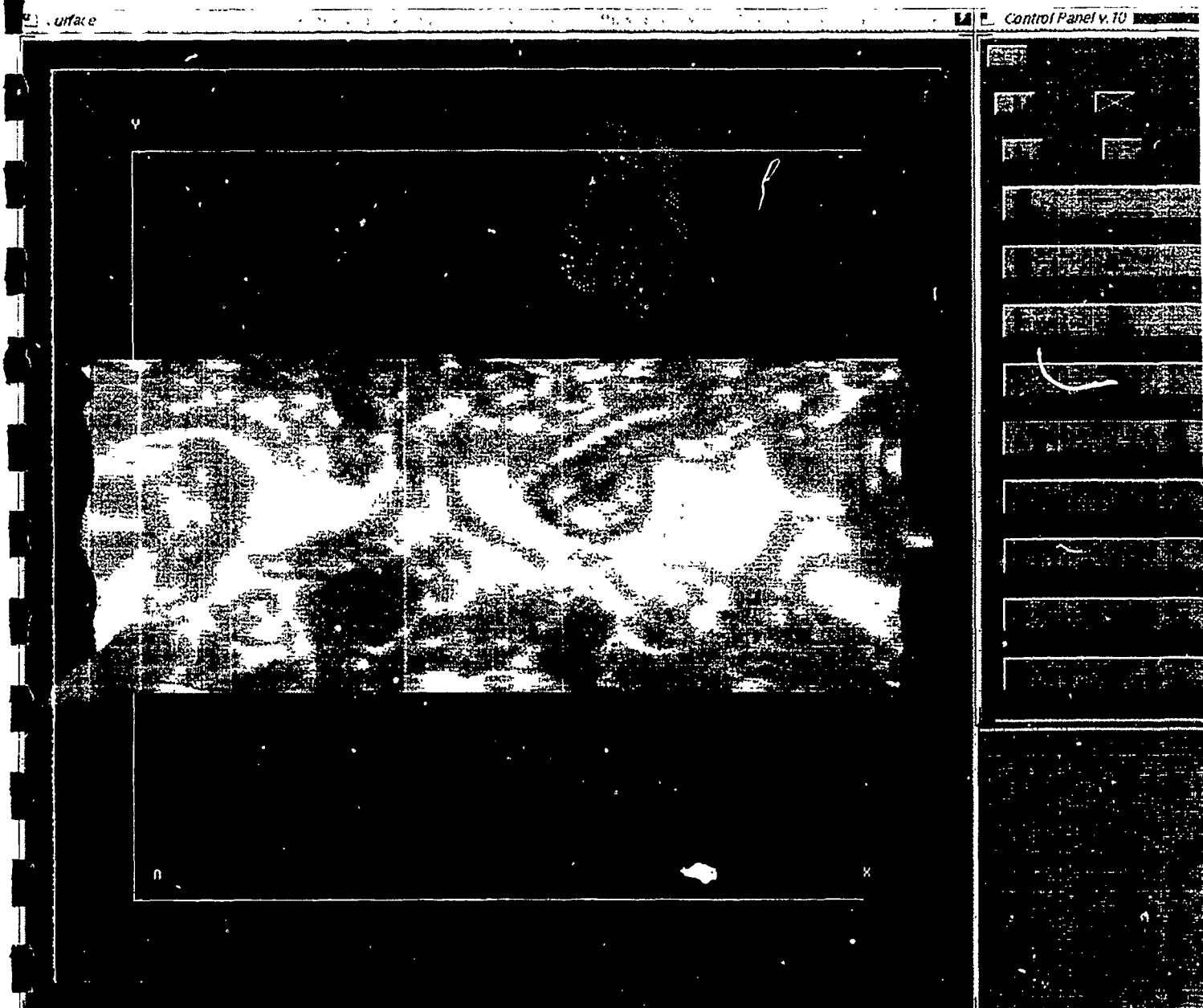
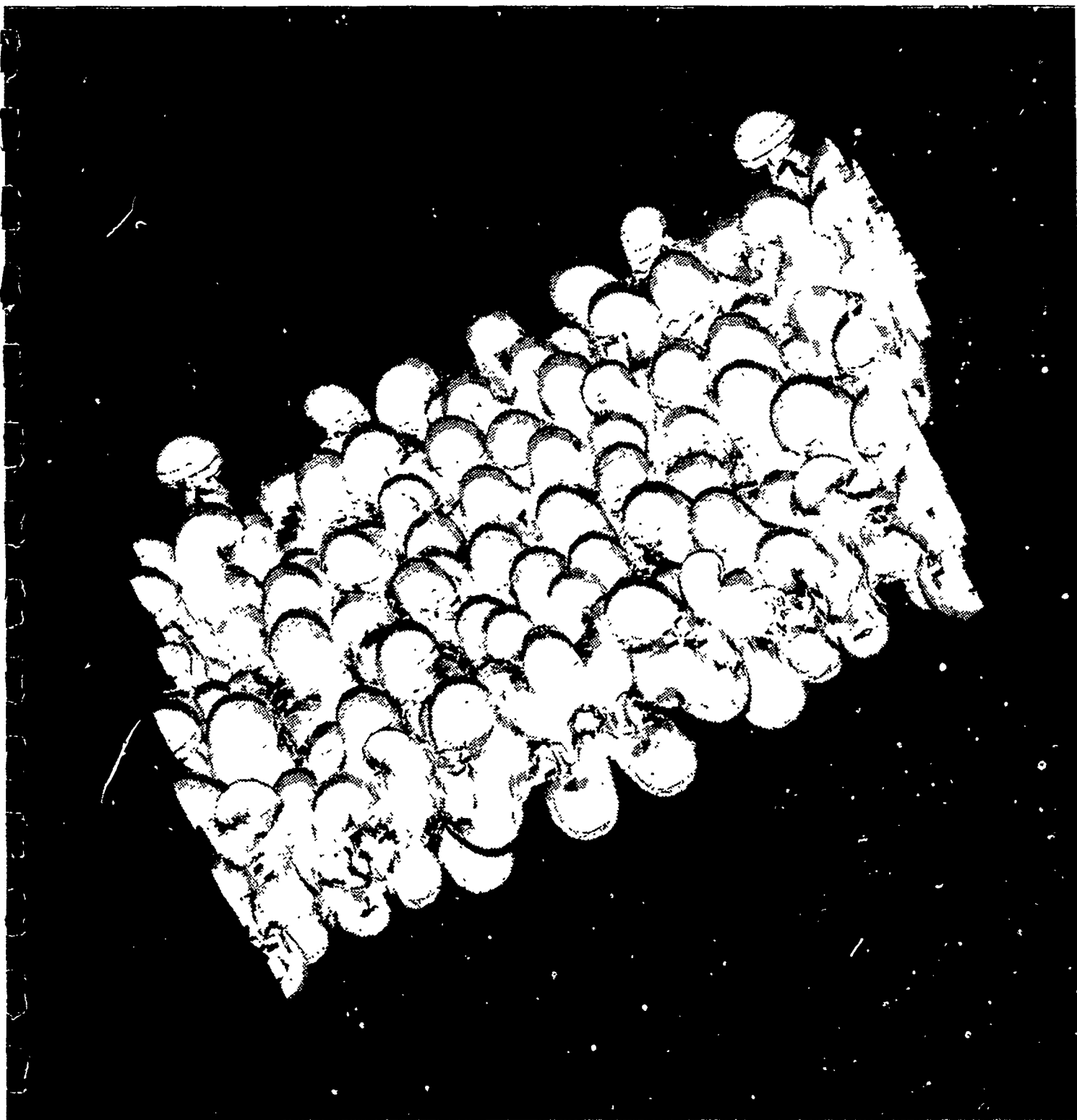


Figure 15: Head-on GEODFORM view
of flame in channel



**Figure 16: Three-dimensional view of
random Rayleigh-Taylor instability**

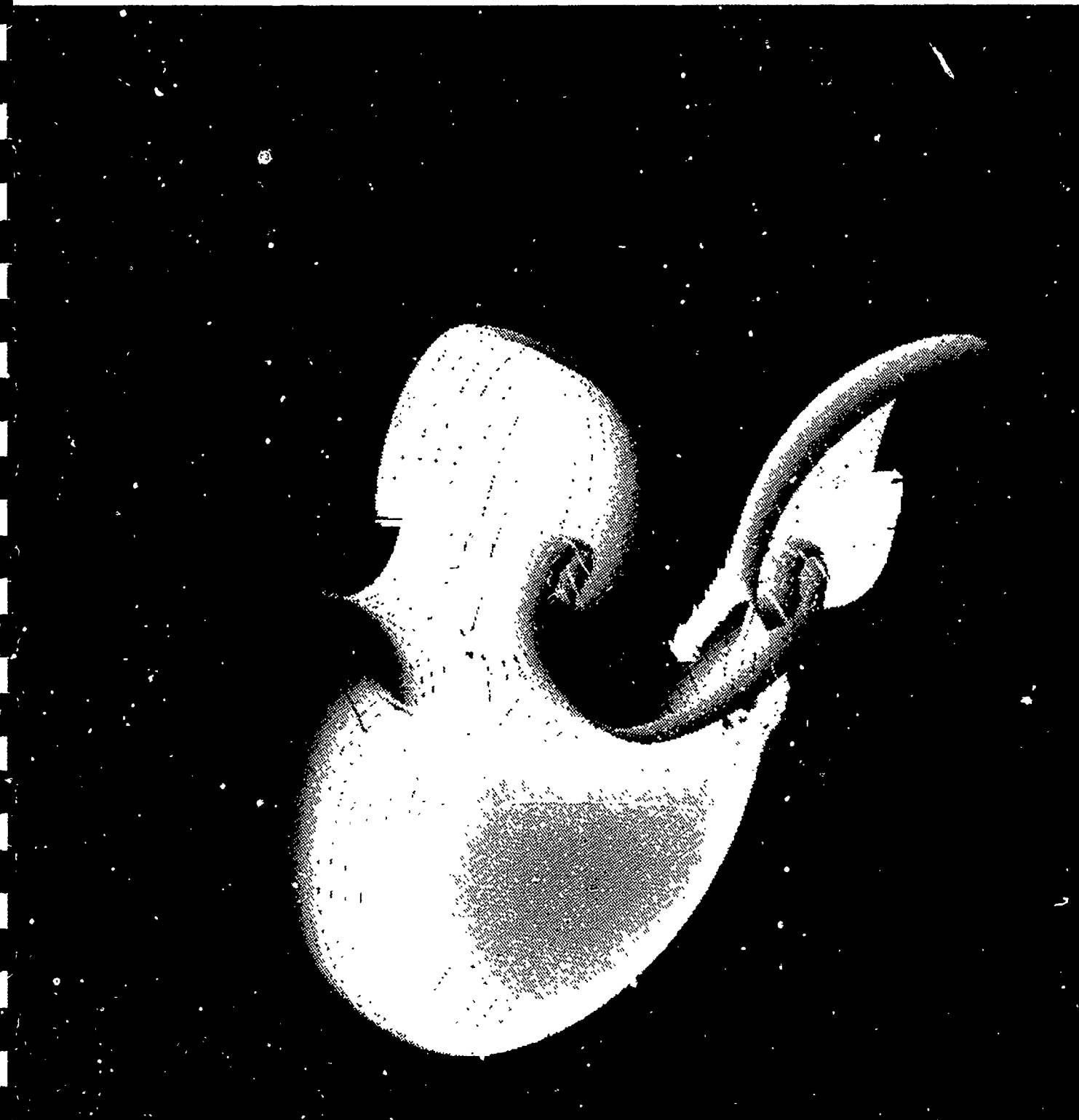


Figure 18: Three-dimensional single frequency Rayleigh-Taylor instability



Figure 19: Early time 3D two-frequency Rayleigh-Taylor instability

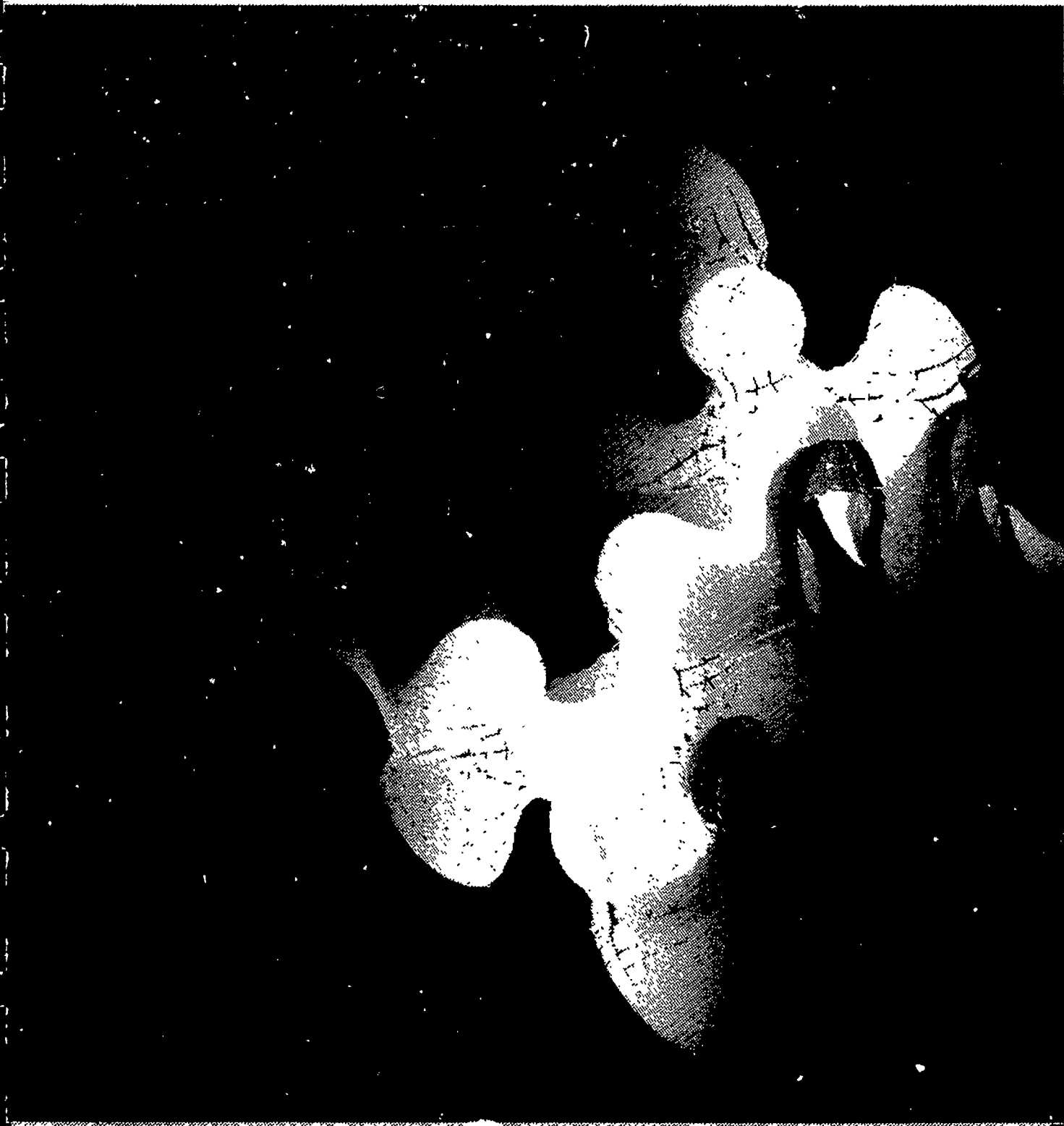


Figure 20: Late time 3D two-frequency Rayleigh-Taylor instability

ccontext.c

```

#include "mclient.n"
#include "gl.h"
#include "stdio.h"
#include "string.h"
#include "device.h"

#define JOB 1
#define EXIT 0

float backvec2[] = {0.0, 0.0, 0.0, 0.8};
float textvec2[] = {1.0, 1.0, 0.0, 0.0};
int mode;
long gld;

try(text)
char text[];
{
    char str[100][100], str1[100];
    float size1, size2;
    FILE *fp, *fopen();
    int i, k;
    fp = fopen(text, "r");
    k = 4;
    mode = JOB;
    i = 0;
    fscanf(fp, "%f %f", &size1, &size2);
    while (fgetc(str1, 100, fp) != NULL) {
        strcpy(str[i], str1);
        ++i;
    }
    foreground();
    preposition(0, 1270, 0, 1000);
    gld = winopen(text);
    winset(gld);
    RGBmode();
    doublebuffer();
    gconfig();
    qdevice(RIGHTMOUSE);

    while(mode) {
        while(!qtest()) {
            tryfont1(str, i, size1, size2);
            swapbuffers();
        }
        prcl(str, i, size1, size2);
    }
    prcl(str, i, size1, size2);
    int i;
    char str[100][100];
    float size1, size2;
    short val;
    int dev;
    while(qtest()) {
        switch(qread(val)) {
            case (RIGHTMOUSE):
                if(val == 1)
                    mode = EXIT;
        }
    }
}

winclose(gld);
break;
case (REDRAW):
    reshapeviewport();
    while(!qtest()) {
        tryfont1(str, i, size1, size2);
        swapbuffers();
    }
    break;
default:
    break;
}
}
}
}

tryfont1(str, i, size1, size2)
int i;
char str[100][100];
float size1, size2;
{
    fmonthandle f;
    fmonthandle fsized;
    int x, y, k, slen, lgld, stepy;
    int beg, stop, start;
    char c;
    c3f(backvec2);
    clear();
    c3f(textvec2);
    if(size1 != 0 & size2 != 0) {
        beg = 1; stop = 2; start = 900;
    }
    else if(size1 != 0) {
        beg = 1; stop = 1; start = 1000 - (1000 - 1 * 100) / 4;
    }
    fminit();
    f = fminfont("Times-Roman");
    fsized = fmscalefont(f, size1);
    fmsfont(fsized);
    for(k = beg; k < stop; k++) {
        x = 100; y = start - k * 150;
        cmov2(x, y);
        fprstr(str[k]);
    }
    stepy = (!<9) ? 100 : (start - (stop * 150)) / 1;
    if(size2 != 0) {
        if(size1 == 0)
            beg = 0;
        else
            beg = 1;
        f = fminfont("Times-Roman");
        fsized = fmscalefont(f, size2);
        fmsfont(fsized);
    }
}

```


ctest.c

```
for (k=begin;k<end;k++)
    x=100; y=700-(k-1)*step;
    cmov2(x,y);
    fprintf(str(k));
    }
```

}

event.c

1

```

/*
 * event.c
 * A more rational way to handle reading the event queue
 * Written by Wade Olsen
 */
#include <stdio.h>
#include <device.h>
#include "event.h"

typedef struct event_s
{
    int window, device, state;
    void (*func)(void *, int);
    char *arg;
    struct event_s *next;
} event_t, *event_p;

typedef struct update_s
{
    int *flag;
    void (*ufunc)(void *);
    char *arg;
    struct update_s *next;
} update_t, *update_p;

static event_p event_list;
static update_p update_list;

/*
 * This routine adds an event to be checked for to the event queue.
 * Window should be the wid of the window to respond in, or ANY if
 * this event applies to all windows. device is the device, and state
 * is the device's value (e.g. ANY, UP, DOWN, the window id (for
 * REDRAW), etc). Func is the function that will be called, with
 * arguments 'arg' and the device's value.
 *
 * NOTE: the device must be queued for it to be found by the event ()
 * routine-- add_event DOES NOT qdevice(device).
 */
void
add_event(window, device, state, func, arg)
int window, device, state;
void (*func)(void *, int);
char *arg;
{
    event_p new_guy;

    new_guy = (event_p)malloc(sizeof(event_t));
    new_guy->window = window;
    new_guy->device = device;
    new_guy->state = state;
    new_guy->func = func;
    new_guy->arg = arg;
    new_guy->next = event_list;
    event_list = new_guy;
}

/*
 * Specify a function to be called if there is nothing in the queue
 * and (*flag) is non-zero. If no update function is active, or
 * (*flag) is 0, then event() will block on a read. If there is an
 */
/*
 * active update function, event() will continuously call the update
 * function, hogging the cpu.
 */
void
add_update(flag, ufunc, arg)
int *flag;
void (*ufunc)(void *);
char *arg;
{
    update_p new_guy;

    new_guy = (update_p)malloc(sizeof(update_t));
    new_guy->flag = flag;
    new_guy->ufunc = ufunc;
    new_guy->arg = arg;
    new_guy->next = update_list;
    update_list = new_guy;
}

/*
 * The main Event. Call this repeatedly to automatically handle
 * reading the queue, and calling your functions to handle what
 * appears there.
 */
void
event()
{
    void find_event(void), event_inputchange(void);
    int find_update(void);
    static int initialized = 0;

    if (initialized == 0)
    {
        add_event(ANY, INPUTCHANGE, ANY, event_inputchange, NULL);
        qdevice(INPUTCHANGE);
        initialized = 1;
    }

    /* Something in the queue? Handle it */
    if (qtest())
        find_event();

    /* Or, if there's no update function, wait for something to appear
     */
    else if (find_update() == 0)
        find_event();
}

int
find_update()
{
    update_p scan;
    int updated = 0;

    for (scan = update_list; scan != 0; scan = scan->next)
    {
        if (*scan->flag)
        {
            (*scan->ufunc)(scan->arg);
            updated = 1;
        }
    }

    return(updated);
}

```

```
event.c
```

```

}

int context, state, device;

void
event_inputchange()
{
    context = winget();
}

void
find_event()
{
    event_p scan;
    short s;

    device = qread(&s);
    state = s;
    for (scan = event_list; scan; scan = scan->next)
    {
        if ( (scan->window == ANY) || (context == scan->window))
            if ( (scan->device == ANY) || (device == scan->device))
                if ( (scan->state == ANY) || (state == scan->state))
                {
                    (*scan->func)(scan->arg, state);
                }
    }
}

```

form_disp.c

1

```
#include <stdio.h>
#include <gl/gl.h>
#include <string.h>
#include <fcntl.h>
#include <math.h>
#include <forms.h>
#include <display.h>
#include <light.h>
#include <cpath.h>

#define maxcol 256
#define MATRIXSIZE 16
#define LIGHTINDEX 10
#define k 3.5
#define ANY -1
#define SIZE 50000

float backvec[] = {0.0, 0.0, 0.0};
float blackvec[] = {0.0, 0.0, 0.0};
float redvec[] = {1.0, 0.0, 0.0};
float whitevec[] = {1.0, 1.0, 1.0};
float yelvec[] = {1.0, 1.0, 0.0};

float color_palette[maxcol][3];

int ncol;
Object Vobj, sphere, Vobj1;
float wx2, wy2, wz2, scal_par;
typedef float Vec[3];

typedef struct {
    int p_nsid;
    Vec ** p;
    int *s;
} Polygon;

int negflag;

float idmat[] = {1.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 1.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 1.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 1.0};

float OldMatrix[16], NewMatrix[16];

Polygon *
MakePolygon(nsid)
int nsid;
{
    Polygon * p;

    p = (Polygon *) malloc (sizeof(Polygon));

    p->p_nsid = nsid;

    p->p = (Vec **) calloc(nsid, sizeof(Vec *));
    p->s = (int *) malloc(3* sizeof(int));

    return p;
}

Vec
*Polygon * P[SIZE];

/*size of matrix in the stack*/
/*size of sun file*/

Vec N[SIZE], N1[SIZE];
int npoints, npolys;
Vec mins = {1000000, 1000000, 1000000};
Vec maxs = {-1000000, -1000000, -1000000};
float V1[SIZE], V2[SIZE][3];

#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

float Max, scal;

float shiftx, shifty, shiftz;

ReadObject()
{
    int i, j, ku, nverts, index[3], nsides;
    Vec tmp1, tmp2;

    scanf("%d %d\n", & npolys, & npoints);

    /*
    fprintf(stderr, "%d points, %d polys\n", npoints, npolys);
    */
    for (j = 0; j < 3; j++) {
        mins[j] = 1e10;
        maxs[j] = -1e10;
    }

    for (i = 0; i < npoints; i++) {
        scanf("%f %f %f\n", &(V[i][0]), &(V[i][1]), &(V[i][2]));
    }

    /*Lala: I changed here */
    for (i = 0; i < npoints; i++) {
        for (ku = 0; ku < 3; ku++) {
            mins[ku] = min(mins[ku], V[i][ku]);
            maxs[ku] = max(maxs[ku], V[i][ku]);
        }
    }

    Max = max(maxs[0]-mins[0], maxs[1]-mins[1]);
    Max = max(Max, maxs[2]-mins[2]);
    scal = ncol / (maxs[2]-mins[2]);
    shiftx = 0.5 * (maxs[0]+mins[0]);
    shifty = 0.5 * (maxs[1]+mins[1]);
    shiftz = 0.5 * (maxs[2]+mins[2]);

    for (i = 0; i < npoints; i++) {
        V[i][0] = V[i][0] - shiftx;
        V[i][1] = V[i][1] - shifty;
        V[i][2] = V[i][2] - shiftz;
    }

    maxs[2] = maxs[2] - shiftz;
    mins[2] = mins[2] - shiftz;

    for (i = 0; i < npolys; i++) {
        P[i] = MakePolygon(3);
        for (j = 0; j < 3; j++) {
            scanf("%d", &index[j]);
            P[i]->s[j] = index[j] - 1;
            P[i]->p[j] = V + (index[j] - 1);
        }
    }
}
```

form_disp.c

2

```

V2[i][j] = V[index[j] - 1][2];
}
V1[i] = V2[i][0] + V2[i][1] + V2[i][2];
V1[i] = V1[i]/3.;
}
/*
fprintf(stderr, "bbx: %f - %f\n", mins[0], maxs[0]);
fprintf(stderr, "bby: %f - %f\n", mins[1], maxs[1]);
fprintf(stderr, "bbz: %f - %f\n", mins[2], maxs[2]);
*/
/*mins[2]=0;*/
scal_par = Max/2.0;

for (i = 0; i < npolys; i++) {
    VecSub(p[i]->p[0], p[i]->p[1], tmp1);
    VecSub(p[i]->p[2], p[i]->p[1], tmp2);
    VecCross(tmp1, tmp2, N(i));
    VecNormalize(N(i));
    if (negflag) {
        if (N(i)[0] < 0.) N(i)[0] = - N(i)[0];
        if (N(i)[1] < 0.) N(i)[1] = - N(i)[1];
        if (N(i)[2] < 0.) N(i)[2] = - N(i)[2];
    }
    else {
        N(i)[0] = - N(i)[0];
        N(i)[1] = - N(i)[1];
        N(i)[2] = - N(i)[2];
    }
}

for (i = 0; i < npolys; i++) {
    N1[i][0] = 0.0;
    N1[i][1] = 0.0;
    N1[i][2] = 0.0;
}

for (i = 0; i < npolys; i++) {
    for (j = 0; j < 3; j++) {
        N1[ P(i)->s[j] ][0] += N(i)[0];
        N1[ P(i)->s[j] ][1] += N(i)[1];
        N1[ P(i)->s[j] ][2] += N(i)[2];
    }
}

for (i = 0; i < npolys; i++)
    VecNormalize(N1[i]);

VecNegate(a)
Vec a;
{
    a[0] = 0 - a[0];
    a[1] = 0 - a[1];
    a[2] = 0 - a[2];
}

VecSub(a, b, c)
Vec a, b, c;
{
    c[0] = a[0] - b[0];
    c[1] = a[1] - b[1];
    c[2] = a[2] - b[2];
}

VecCross(a, b, c)
Vec a, b, c;
{
    c[0] = a[1] * b[2] - b[1] * a[2];
    c[1] = - (a[0] * b[2] - b[0] * a[2]);
    c[2] = a[0] * b[1] - b[0] * a[1];
}

VecNormalize(a)
Vec a;
{
    float l;
    l = (float) sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]);
    a[0] /= l;
    a[1] /= l;
    a[2] /= l;
}

float shinymaterial[] = { SPECULAR, 1.0, 0.2, 0.8,
    EMISSION, 0.8, 0.8, 0.0,
    DIFFUSE, 0.8, 0.0, 0.0,
    AMBIENT, 0.2, 0.0, 0.0,
    SHININESS, 3.0,
    LMNULL };

float sun1[] = { ICOLOR, 1.0, 1.0, 1.0,
    POSITION, 0.0, 0.0, 1.0, 0.0,
    LMNULL };

float sun2[] = { ICOLOR, 1.0, 0.0, 0.0,
    POSITION, 0.0, 0.0, 1.0, 0.0,
    LMNULL };

float sun3[] = { ICOLOR, 0.0, 0.0, 1.0,
    POSITION, 0.0, 0.0, 1.0, 0.0,
    LMNULL };

float kvadrat1[] = {-0.5, -0.5, -0.5};
float kvadrat2[] = {-0.5, 0.5, -0.5};
float kvadrat3[] = {0.5, 0.5, -0.5};
float kvadrat4[] = {0.5, -0.5, -0.5};
Drawbox()
{
    char string1[5], string2[5], string3[5], string4[5];
    sprintf(string1, "%s", "X");
    sprintf(string2, "%s", "Y");
    sprintf(string3, "%s", "Z");
    sprintf(string4, "%s", "O");
    c3f(whitevec);
    cmov(0.50, -0.47, -0.50);
    charstr(string1);
    cmov(-0.50, 0.53, -0.50);
    charstr(string2);
    cmov(-0.53, -0.53, 0.53);
    charstr(string3);
    cmov(-0.47, -0.47, -0.47);
    charstr(string4);
}

```

form_disp.c

3

```

pushmatrix();
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();
pushmatrix();
translate(0.0,0.0,1.0);
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

pushmatrix();
rotate(-90,'y');
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

init_palette()
{
    int i,j;
    float dx1;
    for(i = 0; i < 3; i++)
        for(j = 0; j < ncol; j++)
            color_palette[j][i] = 0.0;

    for(i = 0; i < 3; i++) {
        dx1 = 4./ncol;
        if(i == 0) {
            for(j = ncol; j >= (int) (3*ncol/4)+1; j--) {
                color_palette[j][i] = 1.0;
            }
            for(j = (int) (3*ncol/4); j >= ncol/2; j--) {
                color_palette[j][i] = dx1*(j - (int) (ncol/2));
            }
        }
        if(i == 1) {
            for(j = (int) (ncol); j >= (int) (3*ncol/4); j--) {
                color_palette[j][i] = 1 - dx1*(j - (int) (3*ncol/4));
            }
            for(j = (int) (3*ncol/4); j >= (int) (ncol/4); j--) {
                color_palette[j][i] = 1.0;
            }
            for(j = (int) (ncol/4); j >= 0; j--) {
                color_palette[j][i] = dx1*j;
            }
        }
        if(i == 2) {
            for(j = ncol/2; j >= (int) (ncol/4); j--) {
                color_palette[j][i] = 1.0 - dx1*(j - ncol/4);
            }
            /* printf("%f %d %d\n", color_palette[j][i], i, j); */
        }
    }
}

for(j = (int) (ncol/4); j >= 0; j--) {
    color_palette[j][1] = 1.0;
}

get_palette()
{
    int j;
    FILE *fp, *fopen();
    fp = fopen("color_palette", "r");

    if(fp != NULL) {
        fscanf(fp, "%d", &ncol);
        for(j = 0; j < ncol; j++)
            fscanf(fp, "%f %f %f", &color_palette[j][0], &color_palette[j][1], &color_palette[j][2]);
    }

    fclose(fp);

    scal=ncol/(maxs[2]-mins[2]);
}
else {ncol = maxcol; init_palette();}

get_surface()
{
    FILE *fp, *fopen();

    fp = fopen("lighting", "r");
    if(fp != NULL) {
        fscanf(fp, "%f %f %f", &shinymaterial[1], &shinymaterial[2], &shinymaterial[3]);
        fscanf(fp, "%f %f %f", &shinymaterial[5], &shinymaterial[6], &shinymaterial[7]);
        fscanf(fp, "%f %f %f", &shinymaterial[9], &shinymaterial[10], &shinymaterial[11]);
        fscanf(fp, "%f %f %f", &shinymaterial[13], &shinymaterial[14], &shinymaterial[15]);
    }
    fscanf(fp, "%f %f %f", &sunl[1], &sunl[2], &sunl[3]);
    lndef(DEFMATERIAL, 1, 19, shinymaterial);
    lmbind(MATERIAL, 1);
    lndef(DEFLIGHT, 1, 10, sunl);
    lmbind(LIGHTO, 1);
}
fclose(fp);

save_position()
{
    FILE *fp, *fopen();
    int i;
    fp = fopen("pnl_position", "w");
    for(i = 0; i < MATRIXSIZE; i++)
        fprintf(fp, "%f ", OldMatrix[i]);
    for(i = 0; i < LIGHTINDEX; i++)
        fprintf(fp, "%f ", sunl[i]);
    for(i = 0; i < LIGHTINDEX; i++)
        fprintf(fp, "%f ", sun2[i]);
    for(i = 0; i < LIGHTINDEX; i++)
        fprintf(fp, "%f ", sun3[i]);
    fclose(fp);
}

```

```

    fl_set_slider_bounds(shlx,-1.0 *Max,Max);
    fl_set_slider_bounds(shly,-1.0 *Max,Max);
    fl_set_slider_bounds(shlz,-1.0 *Max,Max);
    fl_set_slider_bounds(xsc,0.1, 10);
    fl_set_slider_bounds(ysc,0.1, 10);
    fl_set_slider_bounds(zsc,0.1, 10);
    fl_set_slider_bounds(size,0.5, 5.0);
}

Set_Slider_Values()
{
    fl_set_slider_value(xrot,0.0);
    fl_set_slider_value(yrot,0.0);
    fl_set_slider_value(zrot,0.0);
    fl_set_slider_value(shlx,0.0);
    fl_set_slider_value(shly,0.0);
    fl_set_slider_value(xsc,1.0);
    fl_set_slider_value(ysc,1.0);
    fl_set_slider_value(zsc,1.0);
    fl_set_slider_value(size,2.0/Max);
}

#define GETVAL(x) fl_get_slider_value(x)

void Change_Stack(FL_OBJECT *obj,long arg)
{
    printf("I got in Change_Stack\n");
    callobj(Vobj);
    if(obj == xrot) rotate((short)GETVAL(xrot) , 'x');
    if(obj == yrot) rotate((short)GETVAL(yrot) , 'y');
    if(obj == zrot) rotate((short)GETVAL(zrot) , 'z');
    if(obj == xsc || obj == ysc || obj == zsc) scale(GETVAL(xsc),GETVAL(ysc),GETVAL(zsc));
    if(obj == size) scale(GETVAL(size),GETVAL(size),GETVAL(size));
    if(obj == shlx || obj == shly || obj == shlz) translate(GETVAL(shlx),
        GETVAL(shly), GETVAL(shlz));
    multmatrix(OldMatrix);
    getmatrix(OldMatrix);
}

MakeVobj()
{
    makeobj(Vobj) = genobj();
    loadmatrix(ldmat);
    ortho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
    closeobj();
}

MakeLight1()
{
    sun1[5] = GETVAL(xlight1);
    sun1[6] = GETVAL(ylight1);
    sun1[7] = GETVAL(zlight1);
    lmdf(DEFLIGHT, 1, 10, sun1);
    lmbind(LIGHT0, 1);
    sun2[5] = GETVAL(xlight2);
    sun2[6] = GETVAL(ylight2);
    sun2[7] = GETVAL(zlight2);
    lmdf(DEFLIGHT, 2, 10, sun2);
    lmbind(LIGHT1, 2);
    sun3[5] = GETVAL(xlight3);
    sun3[6] = GETVAL(ylight3);
}

FILE *fp,*fopen();
fp = fopen("pnl_position","r");
if(fp != NULL) {
    for(i = 0; i < MATRIXSIZE; i++)
        fscanf(fp,"%f",&(OldMatrix[i]));
    for(i = 0; i < LIGHTINDEX; i++)
        fscanf(fp,"%f",&(sun1[i]));
    for(i = 0; i < LIGHTINDEX; i++)
        fscanf(fp,"%f",&(sun2[i]));
    for(i = 0; i < LIGHTINDEX; i++)
        fscanf(fp,"%f",&(sun3[i]));
    fclose(fp);
    lmdf(DEFLIGHT, 1, 10, sun1);
    lmdf(DEFLIGHT, 2, 10, sun2);
    lmdf(DEFLIGHT, 3, 10, sun3);
    if(light1->val) lmbind(LIGHT0, 1); else lmbind(LIGHT0, 0);
    if(light2->val) lmbind(LIGHT1, 2); else lmbind(LIGHT1, 0);
    if(light3->val) lmbind(LIGHT2, 3); else lmbind(LIGHT2, 0);
    loadmatrix(OldMatrix);
}

long lqid;

init_mywin()
{
    keepaspect(1, 1);
    foreground();
    lqid = winopen("surface");
    RGBmode();
    doublebuffer();
    gconf1g();
    lsetdepth(0, 0x7FFFFFFF);
    zbuffer(TRUE);
    mmode(MVIEWING);
}

Set_Slider_Bounds()

l_set_slider_bounds(xrot,-1800,1800);
l_set_slider_bounds(yrot,-1800,1800);
l_set_slider_bounds(zrot,-1800,1800);
}

```

form_disp.c

5

```

sun3[7] = GETVAL(zlight3);
lndef(DEFLIGHT, 3, 10, sun3);
lmbind(LIGHT2, 3);

}

void save_picture(FL_OBJECT *obj, long arg)
{
    char savePict[50], *out_rgb;
    long xor, yor, xsize, ysize;

    winset(lgid);
    getorigin(&xor, &yor);
    getsize(&xsize, &ysize);

    out_rgb = fl_get_input(obj);
    sprintf(savePict, "%s %d %d %d", "scrsave", out_rgb, xor,
        xor+xsize, yor, yor+ysize);
    system(savePict);
}

void make_reset(FL_OBJECT *obj, long arg)
{
    callobj(Vobj1);
    scale(2.0/Max, 2.0/Max, 2.0/Max);
    getmatrix(OldMatrix);
}

void DrawObj(FL_OBJECT *obj, long arg)
{
    static FL_OBJECT *CurObj;

    int i, j, col_ind;
    if(obj == map || obj == model || obj == surface || obj ==
        smooth) CurObj = obj;
    get_surface();
    winset(lgid);
    reshapeviewport();
    c3f(backvec);
    clear();
    zclear();
    MakeVobj();
    callobj(Vobj1);
    MakeLight1;
    if(obj == box) {
        pushmatrix();
        scale(2.0, 2.0, 2.0);
        Drawbox();
        popmatrix();
    }
    if (CurObj == model) {
        mbind(MATERIAL, 0);
        mbind(LIGHT0, 0);
        mbind(LIGHT1, 0);
        mbind(LIGHT2, 0);
        mbind(LMODEL, 0);
        for (i = 0; i < npolys; i++) {
            c3f(redvec);
            bgnclosedline();
            for (j = 0; j < P[i] -> p_nsid; j++) {
                v3f(P[i] -> p[j]);
            }
            endclosedline();
        }
    }
    if (CurObj == map) {
        lmbind(MATERIAL, 0);
        lmbind(LIGHT0, 0);
        lmbind(LIGHT1, 0);
        lmbind(LIGHT2, 0);
        lmbind(LMODEL, 0);
        for (i = 0; i < npolys; i++) {
            bgnpolygon();
            for (j = 0; j < P[i] -> p_nsid; j++) {
                col_ind = (int)((V2[i][j] - mins[2]) * scal);
                col_ind = min(ncol - 1, max(0, col_ind));
                c3f((color_palette[ col_ind][0]));
                v3f(P[i] -> p[j]);
            }
            endpolygon();
        }
    }
    if (CurObj == surface || CurObj == smooth) {
        lmbind(MATERIAL, 1);
        lmbind(LIGHT0, 1);
        lmbind(LIGHT1, 2);
        lmbind(LIGHT2, 3);
        lmbind(LMODEL, 1);
        for (i = 0; i < npolys; i++) {
            bgnpolygon();
            for (j = 0; j < P[i] -> p_nsid; j++) {
                c3f(yelvec);
                if (CurObj == surface) n3f(N[i]);
                if (CurObj == smooth) n3f(N1[P[i] -> s[j]]);
                v3f(P[i] -> p[j]);
            }
            endpolygon();
        }
        swapbuffers();
    }
}

void show_cedit(FL_OBJECT *obj, long arg)
{
    char *str[100];
    sprintf(str, "%s %d %s", PATH, "mycedit", ncol, "t");
    system(str);
}

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fp, *fopen();
    FL_OBJECT *obj;
    int i, j;
    short val;
}

```


form_disp.c

6

```
int dev;
ncol = maxcol;

negflag = 0;
if (argc == 2)
    negflag = 1;
init_palette();
ReadObject();

init_mywin();
create_the_display();
create_the_light();
Set_Slider_Bounds();
Set_Slider_Values();
fl_show_form(DISPLAY, FL_PLACE_SIZE, TRUE, "display");
DrawObj();
while (obj != quit_disp) {
    obj = fl_do_forms();
    DrawObj(obj, 0);
}
```

form_geo.c

1

```
#include<stdio.h>
#include <forms.h>
#include "geomplane.h"
#include "option_form.h"
#include <string.h>
#include "cpath.h"
#include "sandwatch"

#define RUN 1
#define TRIANGULATION 2
#define INTERPOLATION 3
#define DATA_EDITOR 4
#define DISPLAY 5

int norm, peak, hardcopy;
float wlog;
char hardstr[100];
char filename[50], runinfo[50], trianginfo[50], showinfo[50];
char leadersur[50], surfaceinfo[50], intinfo[50], geomview[50];
char text[50], cleanstr[50];
FILE *fp1, *fp2, *fp, *fopen();
int max_size;
int menu2;
int get_max_size();
char stroka[100];

void show_info(FL_OBJECT *obj, long arg) {
    fl_show_message("GEOFORM Version 2.0 ", "Cambridge Hydrodynamics Inc.", "");
}

initialize() {
    printf(runinfo, "%s", PATH, "run.info");
    printf(trianginfo, "%s", PATH, "triang.info");
    printf(showinfo, "%s", PATH, "show.info");
    printf(intinfo, "%s", PATH, "int.info");
    printf(surfaceinfo, "%s", PATH, "surface.info");
    printf(geomview, "%s", PATH, "geomview");
    printf(text, "%s", PATH, "text1");
    printf(leadersur, "%s", PATH, "leader.sur");
    printf(cleanstr, "%s", PATH, "clean");
    system(cleanstr);
    fpl=fopen(leadersur, "r");
    fp2 = fopen("leader", "w");
    while(fgets(stroka, 100, fpl) != NULL)
        fprintf(fp2, "%s", stroka);
    fclose(fpl);
    fclose(fp2);
    wlog = -1;
    peak = -1;
    norm = 0;
    hardcopy = 0;
}

myMenus() {
    menu2 = defpup("ADD POINTS\t %f12*size of data %x2| 4*size of data %x4| 8 *size of data %x8", get_max_size);
}

void choice(FL_OBJECT *obj, long arg) {
    if (arg == RUN) {
        max_size = dopup(menu2);
        change_cursor();
        start_(filename, &norm, &wlog, &peak);
        trian_(&hardcopy);
        if (max_size > 1) {
            add_(&max_size);
            int_(&max_size);
            if (hardcopy) system(hardstr);
        }
        restore_cursor();
        fl_show_message("DATA IS PROCESSED", "PRESS SAVE BUTTON ", "TO SAVE OUTPUT IN THE FILE");
    }
    if (arg == TRIANGULATION) {
        change_cursor();
        start_(filename, &norm, &wlog, &peak);
        trian_(&hardcopy);
        restore_cursor();
        fl_show_message("TRIANGULATION IS DONE", "", "");
    }
    if (arg == INTERPOLATION) {
        max_size = dopup(menu2);
        if ((fpl = fopen("triang", "r")) == NULL) {
            fl_show_message("YOU HAVE TO RUN", "TRIANGULATION", "FIRST");
            fclose(fpl);
        }
        else {
            change_cursor();
            add_(&max_size);
            int_(&max_size);
            if (hardcopy) system(hardstr);
            restore_cursor();
            fl_show_message("INTERPOLATION", "IS ", "DONE");
        }
    }
    if (arg == DATA_EDITOR) {
        start_(filename, &norm, &wlog, &peak);
        show_(text, &hardcopy);
        if (hardcopy) system(hardstr);
    }
    if (arg == DISPLAY) {
        change_cursor();
        system(geomview);
        printf("%s\n", geomview);
        restore_cursor();
    }
}
```

form_geo.c

2

```

void save_triang(FL_OBJECT *obj, long arg)
{
    char locstr[100];
    sprintf(locstr, "%s", "launch -h \"cp triang\" -m \"Save file as\" -c \"",
            filename, ".tri\"");
    system (locstr);
}

void get_menu(FL_OBJECT *obj, long arg)
{
    fl_show_form(option_form, FL_PLACE_SIZE, FALSE, NULL);
}

int quit = 0;
void make_quit(FL_OBJECT *obj, long arg)
{
    quit = 1;
}

void normalize(FL_OBJECT *obj, long arg)
{
    norm = 1;
}

void logar(FL_OBJECT *obj, long arg)
{
    wlog = 8;
}

void kill_peak(FL_OBJECT *obj, long arg)
{
    peak = fl_show_question("ERRORS IN THE FILE?", "", "");
}

void reset(FL_OBJECT *obj, long arg)
{
    wlog = -1;
    peak = -1;
    norm = 0;
    hardcopy = 0;
}

set_max_size(n)
int n;
{
    ax_size = n;
    return n;
}

void show_files(FL_OBJECT *obj, long arg)
{
    char stroka[100], *fname;
    system(clearstr);
    fname = fl_show_file_selector("INPUT FILE NAME", "", "", "");
}

if(fname == NULL || fname[0] == '\0') return;
else
    strcpy(filename, fname);

fp1 = fopen(leader, "r");
fp2 = fopen("leader", "w");

while(fgets(stroka, 100, fp1) != NULL) {
    fprintf(fp2, "%s", stroka);
}
fclose(fp1);
fclose(fp2);
strcpy(fnameobj->label, filename);
fl_redraw_object(fnameobj);
}

make_hardcopy()
{
    hardcopy = 1;
    sprintf(hardstr, "%s",
            "launch -h \"mv hardtmp\" -m \"Save hardcopy as\" -c \"",
            filename, ".mongo\"");
}

void get_out( FL_OBJECT *obj, long arg)
{
    fl_hide_form(option_form);
}

change_cursor()
{
    curstype(C16X1);
    defcursor(1, sandwatch_bits);
    curorigin(1, 8, 8);
    setcursor(1, 0, 0);
}

restore_cursor()
{
    setcursor(0, 0, 0);
}

main(argc, argv)
int argc;
char *argv[];
{
    FL_OBJECT *obj;
    char *fname;
    if(argc < 2) {
        fname = fl_show_file_selector("INPUT FILE NAME", "", "", "");
        if(fname != NULL && fname[0] != '\0')
            sprintf(filename, "%s", fname);
    }
    else
        strcpy(filename, argv[1]);
}

```

form_geo.c

3

```
initialize();
create_the_forms();
create_the_options();
strcpy(fnameobj->label, filename);
fl_redraw_object(fnameobj);
fl_show_form(GEOPLANE, FL_PLACE_SIZE, TRUE, NULL);
    myMenus();
while (obj != exit_but)
obj = fl_do_forms();
}
fl_hide_form(GEOPLANE);
system(cleanstr);
}
```

ftext.c

```

#include "fcntl.h"
#include "gl.h"
#include "stdio.h"
#include <string.h>
#include <device.h>

#define JOB 1
#define EXIT 0

float backvec3[]={0.0,0.0,0.8};
float textvec3[]={1.0,1.0,0.0};
int mode;
long gld;

tryl_(text)
char text[];
{
    char str[100][100],strl[100];
    float szel,size2;
    FILE *fp,*fopen();
    int l,k;
    fp = fopen(text,"r");
    k = 4;
    mode = JOB;
    l=0;
    fscanf(fp,"%f %f %f",&szel,&size2);
    while (fgetc(strl,100,fp)!= NULL) {
        strcpy(str[l],strl);
        ++l;
    }
    foreground();
    prefposition(0,1270,0,1000);
    gld = winopen(text);
    winset(gld);
    RGBmode();
    doublebuffer();
    qconf1g();
    c3f(backvec3);
    clear();
    qdevice(RIGHTHOUSE);

    while(mode) {
        while(!qtest()) {
            tryfont1(str,l,szel,size2);
            swapbuffers();
        }
        prol(str,l,szel,size2);
    }
}

```

geoform.c

```

#include<stdio.h>
#include<device.h>
#include<fcntl.h>
#include<gl.h>
#include<string.h>
#include<math.h>

#define sla 0.35
#define xla 0.325
#define yla 0.45

#define INFO 1
#define GO 2

float compas[16][2] = {
    {xla +sla/2, yla},
    {xla+sla *3/8, yla+sla/2},
    {xla+sla/2, yla+sla},
    {xla+sla *5/8, yla+sla/2},
    {xla, yla+sla/2},
    {xla+sla/2, yla+sla *3/8},
    {xla+sla, yla+sla/2},
    {xla+sla/2, yla+sla *5/8},
    {xla+sla/4, yla+sla *3/4},
    {xla+sla/4, yla+sla/4},
    {xla+sla *3/4, yla+sla/4},
    {xla+sla *3/4, yla+sla *3/4},
    ;

float whitevec2[] = {1.0,1.0,1.0};
float whitevec[] = {0.5,0.8,1.0};
float backvec[] = {0.0,1.0,0.0};
float bluevec[] = {0.0,0.0,1.0};
float yelvec[] = {1.0,1.0,0.0};
float redvec[] = {1.0,0.0,0.0};
ong xo,yo,xsize,ysize;
ong lgid, lgid1;
nt menu,menu1,menu2,submenu1,submenu2,submenu3,submenu4,submenu5;
.hort mx,my;
nt mode, job, exitmode;
har filename[30],runinfo[50],trianginfo[50],showinfo[50],leadersur[50];
har surfaceinfo[50],intinfo[50],geomview[50],text[50];
Int max_size;

main(argc,argv)
Int argc;
char *argv[];
{
    char *stroka[100],name[100] ;
    Int i;

    FILE *fp1,*fp2,*fopen();
    if (argc< 3)
        printf("usage: geoplane [name of file] [number of points after interpolation]\n");
    else {
        printf(runinfo,"%s",PATH,"run.info");
        printf(trianginfo,"%s",PATH,"triang.info");
        printf(showinfo,"%s",PATH,"show.info");
        printf(intinfo,"%s",PATH,"int.info");
        printf(surfaceinfo,"%s",PATH,"surface.info");
    }
}
printf(geomview,"%s",PATH,"geomview",PLACE);
printf(text,"%s",PATH,"text");
printf(leadersur,"%s",PATH,"leader.sur");
max_size = atoi(argv[2]);
fp1=fopen(leadersur,"r");
fp2 = fopen("leader","w");
i=1;
while(fgets(stroka,100,fp1)!=NULL) {
    if(i == 20) {
        if(max_size != 0)
        {
            fprintf(fp2,"%d\n",max_size);
        }
        else {
            sscanf(stroka,"%d",&max_size);
            fprintf(fp2,"%s",stroka);
        }
    }
    else {
        fprintf(fp2,"%s",stroka);
        i++;
    }
    fclose(fp1);
    fclose(fp2);
    strcpy(filename,argv[1]);
    initialise();
    concave(TRUE);
    Drawobj();

    while(!exitmode) {
        while(!qtest()) {
            Drawobj();
            swapbuffers();
        }
        processinput();
    }
    winclose(lgid);
    system("clean");
}

initialise()
{
    job = -1;
    exitmode = 0;
    keepaspect(1,1);
    foreground(0);
    prefsize(700,700);
    lgid = winopen("geoplane");
    RGmode();
    doublebuffer();
    qconf();
    ortho(0.0,1.0,0.0,1.0,-1.0,1.0);
    c3f(backvec);
    clear();
    qdevice(RIGHTMOUSE);

    tie(RIGHTMOUSE,MOUSEX,MOUSEY);
    getorigin(&xo,&yo);
    getsize(&xsize,&ysize);
    myMenu();
}

```

```

{
    printf("kill is working %d \n,1");
}

smooth(i)
int i;
{
    printf("smooth is working %d \n,1");
}

Drawobj()
{
    fmonthandle f;
    fmonthandle fsized;

    int i,k,ind,angle;
    float x0,y0,xre,yre,size,rad,radi;
    char *str[15],strl[100],stril[100],strir[100];
    char string[50],st[100];
    str[0] = "lotion";
    str[1] = "triangu";
    str[2] = "geoplane";
    str[3] = "run";
    str[4] = "lotion";
    str[5] = "interpo";
    str[6] = "options";
    str[7] = " ";
    str[8] = "change";
    str[9] = "data ";
    str[10] = "the model";
    str[11] = "view ";

    size = 24.0;
    rad = 0.1;
    radi = 0.09;
    printf(strl," %s %d ",filename,max_size);
    sprintf(stril," GEOPLANE ");
    fminit();
    f = fminifont("Times-Roman");
    fsized = fmscalefont(f,size);
    fmsfont(f,sized);

    c3f(backvec);
    rectf(0.0,0.0,1.0,1.0);
    c3f(whitevec);
    rectf(0.3,0.05,0.7,0.15);
    cmov2(0.35,0.095);
    c3f(bluevec);
    fprstr((unsigned char *)stril);
    rectf(0.2,0.85,0.7,0.95);
    c3f(yelvevec);
    cmov2(0.34,0.89);
    fprstr((unsigned char *)stril);

    x0 = 0.1; y0 = 0.2;
    ind = 0;
    for(i = 0; i < 3; i++) {
        for(k = 0; k < 2; k++) {
            xre = x0 + i * 0.3 + rad;
            yre = y0 + k * 0.4 + rad;

```

```

c3f(whitevec2);
rectf(xre-rad,yre-rad,xre+rad,yre+rad);
if(ind != job)
c3f(bluevec);
else c3f(redvec);
rectf(xre-rad1,yre-rad1,xre+rad1,yre+rad1);
ind++;
}
}
ind = 0;
for(i = 0; i < 3; i++) {
    for(k = 0; k < 2; k++) {
        xre = x0+i*0.3;
        yre = y0+k*0.4;
        c3f(yelvec);
        cmov2(xre+0.02,yre+0.07);
        fmrstr((unsigned char *) str[ind]);
        cmov2(xre+0.03,yre+0.13);
        fmrstr((unsigned char *) str[ind+1]);
        ind = ind+2;
    }
}
}

/*picture();*/
}

processInput()
{
    short val;
    int dev, i, k, m;
    long menuval;
    int k = m = 0;
}

```

```

choice()
{
    float xco,yco;
    int ind1,ind2,ind5;
    xco = (float)(mx -xo)/(float)xsize;
    yco = (float)(my -yo)/(float)ysize;
    ind1 = 1;
    ind2 = 0;
    if((xco<0.3)&&(xco>0.1)){
        if(yco<0.4&&yco>0.2)
        {
            if(mode == GO)
            {
                start_(filename,&ind2);
                trian_(&ind1,&ind5);
                job = -1;
            }
            if (mode == INFO)
            try(trianginfo);
            job = -1;
        }
        else if (yco>0.6&&yco<0.8)
        {
            if(mode == GO)
            {
                ind1 = 0;
                start_(filename,&ind2);
                job = 0;
            }
        }
    }
}

```


geoform.c

4

```

Drawobj(); swapbuffers();
triang(&ind2,&ind5);
if(ind5 == 1){
Drawobj();
swapbuffers();
}
else
job = 2;
Drawobj(); swapbuffers();
int_(&ind2);
job = 5;
Drawobj(); swapbuffers();
system("rm tempor actual text");
sleep(10);
job = -1;
}
if (mode == INFO)
try(runinfo);
job = -1;
}
}

else if ((xco<0.6)&&(xco>0.4)){
if(yco<0.4&&yco>0.2)
{
if(mode == GO){
int_(&ind1);
system("rm rescales tempor actual");
job = -1;
}
if (mode == INFO){
try(intinfo);
job = -1;
}
}
else if(yco>0.6&&yco<0.8)
{
system("not leader &");
job = -1;
}
}

else if((xco<0.9)&&(xco>0.7))
if(yco>0.2&&yco<0.4){
if(mode == GO){
start_(&llname,&ind2);
show(text);
system("rm rescales");
job = -1;
}
if(mode == INFO){
try(showinfo);
job = -1;
}
}
else if(yco>0.6&&yco<0.8)
{
if(mode == GO){
system("geomview<triang &");
sleep(10);
}
}
}

job = -1;
if(mode == INFO){
try(surfaceinfo);
job = -1;
}
}
}

choicel()
{
float xco,yco;
xco = (float)(mx -xo)/(float)xsize;
yco = (float)(my -yo)/(float)yssize;
if((xco<0.3)&&(xco>0.1)){
if(yco<0.4&&yco>0.2)
job = 0;
else if (yco>0.6&&yco<0.8)
job = 1;
}
}

else if ((xco<0.6)&&(xco>0.4)){
if(yco<0.4&&yco>0.2)
job = 2;
else if(yco>0.6&&yco<0.8)
job = 3;
}
}
else if(xco<0.9&&xco>0.7){
if(yco<0.4&&yco>0.2)
job = 4;
else if (yco>0.6&&yco<0.8)
job = 5;
}
}

if(xco<0.7 && xco>0.3)
{
if(yco>0.85 && yco<0.95)
job = -2;
}
}

picture()
{
concave(TRUE);
c3f(bluevec);
bgnpolygon();
v2f(&compas[8][0]);
v2f(&compas[11][0]);
}

```

geoform.c

5

```

v2f(&compas[9][0]);
v2f(&compas[5][0]);
v2f(&compas[10][0]);
v2f(&compas[3][0]);
v2f(&compas[11][0]);
v2f(&compas[7][0]);
v2f(&compas[8][0]);
endpolygon();
c3f(yelvec);
bgnclosedline();
v2f(&compas[8][0]);
v2f(&compas[11][0]);
v2f(&compas[9][0]);
v2f(&compas[5][0]);
v2f(&compas[10][0]);
v2f(&compas[3][0]);
v2f(&compas[7][0]);
v2f(&compas[11][0]);
v2f(&compas[8][0]);
endclosedline();
move2(compas[9][0],compas[9][1]);
draw2(compas[11][0],compas[11][1]);
move2(compas[8][0],compas[8][1]);
draw2(compas[10][0],compas[10][1]);
c3f(bluevec);
bgnpolygon();
v2f(&compas[4][0]);
v2f(&compas[5][0]);
v2f(&compas[6][0]);
v2f(&compas[7][0]);
v2f(&compas[4][0]);
endpolygon();
c3f(yelvec);
bgnclosedline();
v2f(&compas[4][0]);
v2f(&compas[5][0]);
v2f(&compas[6][0]);
v2f(&compas[7][0]);
v2f(&compas[4][0]);
endclosedline();
c3f(bluevec);
bgnpolygon();
v2f(&compas[0][0]);
v2f(&compas[11][0]);
v2f(&compas[2][0]);
v2f(&compas[3][0]);
v2f(&compas[0][0]);
endpolygon();
c3f(yelvec);
bgnclosedline();
v2f(&compas[0][0]);
v2f(&compas[11][0]);
v2f(&compas[2][0]);
v2f(&compas[3][0]);
v2f(&compas[0][0]);
endclosedline();

c3f(yelvec);
move2(compas[0][0],compas[0][1]);
draw2(compas[2][0],compas[2][1]);
move2(compas[4][0],compas[4][1]);
draw2(compas[11][0],compas[11][1]);
move2(compas[3][0],compas[3][1]);

```

```

draw2(compas[6][0],compas[6][1]);

```

```

)

```

geoform1.c

```

#include<stdio.h>
#include<device.h>
#include<fmclient.h>
#include<gl.h>
#include<string.h>
#include<cpath.h>

#define sla 0.20
#define xla 0.005
#define yla 0.805

#define INFO 1
#define GO 2

float compas[16][2] = {
    {xla+sla/2,yla},
    {xla+sla*3/8,yla+sla/2},
    {xla+sla/2,yla+sla},
    {xla+sla*5/8,yla+sla/2},
    {xla,yla+sla/2},
    {xla+sla/2,yla+sla*3/8},
    {xla+sla,yla+sla/2},
    {xla+sla/2,yla+sla*5/8},
    {xla+sla/4,yla+sla*3/4},
    {xla+sla/4,yla+sla/4},
    {xla+sla*3/4,yla+sla/4},
    {xla+sla*3/4,yla+sla*3/4}
};

float whitevec2[] = {1.0,1.0,1.0};
float whitevec1[] = {0.5,0.8,1.0};
float backvec1[] = {0.0,1.0,0.0};
float bluevec[] = {0.0,0.0,1.0};
float yelvec[] = {1.0,1.0,0.0};
float redvec[] = {1.0,0.0,0.0};
long xo,yo,xsize,ysize;
long lgld, lgldi;
short mx,my;
int norm, peak, hardcopy;
float wlog;
char hardstr[100];
int mode, job, exitmode, getout;
char filename[50], runinfo[50], trianginfo[50], lea_ersur[50];
char surfacinfo[50], intinfo[50], geomview[50], text[50], prepar[50], clean[50];
int max_size;
main(argc,argv)
int argc;
char *argv[];
{
    char stroka[100],name[100] ;
    int i;

    FILE *fp1,*fp2,*fp,*fopen();
    sprintf(runinfo,"%s",PATH,"run.info");
    sprintf(trianginfo,"%s",PATH,"triang.info");
    sprintf(showinfo,"%s",PATH,"show.info");
    sprintf(intinfo,"%s",PATH,"int.info");
    sprintf(surfacinfo,"%s",PATH,"surface.info");
    sprintf(geomview,"%s",PATH,"geomview");
    sprintf(text,"%s",PATH,"text1");

```

geoform1.c

2

```

char locstr[100];
sprintf(locstr,"%s%s","launch -h \"cp triang\" -m \"Save file as\" -c \"",
        filename,".tri\\\"");
system (locstr);
}

myMenu ()
{
    menu = defpup("GEOPLANE %t\\INFO igo ");
    menu1 = defpup("GEOPLANE %t");
    addtopup(menu1," OPEN %f ",restart);
    addtopup(menu1," SAVE %f! QUIT ",save_triang);
    menu2 = defpup("Level of smoothing %f! 2 %x2! 4 %x4! 8 %x8",get_max_size);
    menu7 = defpup("LOGARITHM %t %f! 2 %x2! 4 %x4! 8 %x8",logar);
    menu8 = defpup("ERRORS IN THE FILE? %t %f! YES|NO",peaks);
    menu9 = defpup("SMOOTH %t, KILL PEAKS %m",menu8);
    addtopup(menu9,"LOGARITHM %f",logar);
    menu10 = defpup("OPTIONS %t NORMALIZE %f",normalize);
    addtopup(menu10,"SMOOTH %m",menu9);
    addtopup(menu10,"HARDCOPY %f",make_hardcopy);
    addtopup(menu10,"RESET ALL %f",reset);
}

Drawobj()
{
    fmfonthandle f;
    fmfonthandle fsized;

    int l,k,ind,angle,len;
    float x0,y0,xre,yre,size,rad,radi;
    char *str1[15],str1[100],str1[100],str12[100];
    char string[50],st[100];
    str1[0] = "lotion";
    str1[1] = "triangu-";
    str1[2] = "geoplane";
    str1[3] = " run";
    str1[4] = "lotion";
    str1[5] = "interpo-";
    str1[6] = "options";
    str1[7] = " ";
    str1[8] = "change";
    str1[9] = " data ";
    str1[10] = "the model";
    str1[11] = " view";

    size = 24.0;
    rad = 0.1;
    radi = 0.09;
    sprintf(str1," %s ",filename);
    sprintf(str1," GEOPLANE ");

    fmfonthandle f;
    f = fmfonthandle("Times-Roman");
    fsize = fmfonthandle(f,size);
    fmsizefont(fsize);

    c3f(backvect);
    rectf(0.0,0.0,1.0,1.0);
}

```

```

c3f(bluevec);
rectf(0.3,0.85,0.7,0.95);
c3f(yelvec);
cmov2(0.34,0.89);
fmrstr(str1);

x0 = 0.1;y0 = 0.2;
ind = 0;
for(l = 0;l<3;l++) {
    for(k = 0;k<2;k++) {
        xre = x0+l*0.3*rad;
        yre = y0+k*0.4*rad;
        c3f(whitevec2);
        rectf(xre-rad,yre-rad,xre+rad,yre+rad);
        if(lnd != job)
            c3f(bluevec);
        else c3f(redvec);
        rectf(xre-rad1,yre-rad1,xre+rad1,yre+rad1);
        ind++;
    }
}
ind = 0;
for(i = 0;i<3;i++) {
    for(k = 0;k< 2; k++) {
        xre = x0+l*0.3;
        yre = y0+k*0.4;
        c3f(yelvec);
        cmov2(xre+0.02,yre+0.07);
        fmrstr(str1[ind]);
        cmov2(xre+0.03,yre+0.13);
        fmrstr(str1[ind+1]);
        ind = ind+2;
    }
}

len = (strlen(filename) + 4)*0.7;
size = (0.4*xsize)/(float)len;
if(size>24)size = 24;
fsize = fmscalefont(f,size);
fmsctfont(fsize);

c3f(whitevec2);
rectf(0.3,0.05,0.7,0.15);
cmov2(0.35,0.035);
c3f(bluevec);
fmrstr(str1);

picture();
,

processInput()
{
    short val;
    int dev,l,k,m;
    long menuval;
    i = k = m = 0;

```

```

choice()
{
    float xco,yco;
    int ind1,ind2,ind5;
    long menuval1;
    xco = (float)(mx -xo)/(float)xsize;
    yco = (float)(my -yo)/(float)ysize;
    ind1 = 1;
    ind2 = 0;
    if((xco<0.3)&&(xco>0.1)){
        if(yco<0.4&&yco>0.2)
        {
            if(mode == GO)
            {
                start_(filename,&norm,&wlog,&peak);
                trian_(&hardcopy);
                if(hardcopy)system(hardstr);
                job = -1;
            }
            if (mode == INFO)
                try(trianginfo);
            job = -1;
        }
        else if (yco>0.6&&yco<0.8)
        {
            if(mode == GO) {
                ind1 = 0;
                max_size = dopup(menu2);
                start_(filename,&norm,&wlog,&peak);
                job = 0;
                Drawobj(); swapbuffers();
                trian_(&hardcopy);
                if(max_size > 1){
                    job = 2;
                    Drawobj(); swapbuffers();
                    add_(&max_size);
                    int_(&max_size);
                    if(hardcopy)system(hardstr);
                }
                job = 5;
                Drawobj(); swapbuffers();
                system("rm tempor actual text");
                system (geomview);
                sleep(10);
                job = -1;
            }
            if (mode == INFO)
                try(runinfo);
            job = -1;
        }
    }
    else if ((xco<0.6)&&(xco>0.4)) {
        if(yco<0.4&&yco>0.2)
        {
            if(mode == GO) {
                max_size = dopup(menu2);
                if(max_size>1){
                    add_(&max_size);
                    Drawobj(); swapbuffers();
                }
            }
            else if ((xco<0.9)&&(xco>0.7))
            if(yco>0.2&&yco<0.4) {
                if(mode == GO) {
                    start_(filename,&norm,&wlog,&peak);
                    show_(text,&hardcopy);
                    if(hardcopy)system(hardstr);
                    system("rm rescales");
                    job = -1;
                }
                if(mode == INFO) {
                    try(showinfo);
                    job = -1;
                }
            }
            else if (yco>0.6&&yco<0.8)
            {
                if(mode == GO) {
                    system(geomview);
                    sleep(10);
                    job = -1;
                }
                if(mode == INFO) {
                    try(surfaceinfo);
                    job = -1;
                }
            }
        }
        float xco,yco;
        xco = (float)(mx -xo)/(float)xsize;
        yco = (float)(my -yo)/(float)ysize;
        if((xco<0.3)&&(xco>0.1)){
            if(yco<0.4&&yco>0.2)
            {
                job = 0;
            }
            else if (yco>0.6&&yco<0.8)
            {
                job = 1;
            }
        }
        else if ((xco<0.6)&&(xco>0.4)) {
            if(yco<0.4&&yco>0.2)
            {
                job = 2;
            }
            else if(yco>0.6&&yco<0.8)
            {
                job = 3;
            }
        }
    }
}

```

geoform1.c

5

```

else if (xco<0.944xco>0.7){
    if (yco<0.444yco>0.2)
        job = 4;
    else if (yco>0.644yco<0.8)
        job = 5;
}

if (xco<0.744 xco>0.3)
{
    if (yco>0.8544 yco<0.95)
        job = -2;
}

v2f (&compas[4][0]);
endpolygon();
c3f (yelvec);
bgnclosedline();
v2f (&compas[4][0]);
v2f (&compas[5][0]);
v2f (&compas[6][0]);
v2f (&compas[7][0]);
v2f (&compas[4][0]);
endclosedline();
c3f (bluevec);
bgnpolygon();
v2f (&compas[0][0]);
v2f (&compas[1][0]);
v2f (&compas[2][0]);
v2f (&compas[3][0]);
v2f (&compas[0][0]);
endpolygon();
c3f (yelvec);
bgnclosedline();
v2f (&compas[0][0]);
v2f (&compas[1][0]);
v2f (&compas[2][0]);
v2f (&compas[3][0]);
v2f (&compas[0][0]);
endclosedline();

```

picture ()

```

concave (TRUE);
c3f (bluevec);
gnpolygon();
v2f (&compas[8][0]);
v2f (&compas[1][0]);
v2f (&compas[9][0]);
v2f (&compas[5][0]);
v2f (&compas[10][0]);
v2f (&compas[3][0]);
v2f (&compas[11][0]);
v2f (&compas[7][0]);
v2f (&compas[8][0]);
endpolygon();
c3f (yelvec);
bgnclosedline();
v2f (&compas[8][0]);
v2f (&compas[9][0]);
v2f (&compas[10][0]);
v2f (&compas[5][0]);
v2f (&compas[10][0]);
v2f (&compas[3][0]);
v2f (&compas[11][0]);
v2f (&compas[7][0]);
v2f (&compas[8][0]);
endclosedline();
ove2 (compas[9][0], compas[9][1]);
raw2 (compas[11][0], compas[11][1]);
ove2 (compas[8][0], compas[8][1]);
raw2 (compas[10][0], compas[10][1]);
c3f (bluevec);
gnpolygon();
v2f (&compas[4][0]);
v2f (&compas[5][0]);
v2f (&compas[6][0]);
v2f (&compas[7][0]);

```

geofrm2.c

```

#include<stdio.h>
#include<device.h>
#include<fcntl.h>
#include<gl.h>
#include<string.h>
#include<cpath.h>

#define sla 0.35
#define xla 0.325
#define yla 0.45

#define INFO 1
#define GO 2

float compas[16][2] = {
    {(xla +sla/2),yla},
    {xla+(sla *3)/8,yla+sla/2},
    {xla+sla/2,yla+sla},
    {xla+(sla *5)/8,yla+sla/2},
    {xla,yla+sla/2},
    {xla+sla/2,yla+(sla *3)/8},
    {xla+sla,yla+sla/2},
    {xla+sla/2,yla+(sla *5)/8},
    {xla+(sla/4),yla+(sla *3)/4},
    {xla+(sla/4),yla+(sla/4)},
    {xla+(sla *3)/4,yla+sla/4},
    {xla+(sla *3)/4,yla+(sla*3)/4}
};

float whitevec2[] = {1.0,1.0,1.0};
float whitevec1[] = {0.5,0.8,1.0};
float backvec1[] = {1.0,1.0,0.0};
float bluevec[] = {0.0,0.1,0.0};
float yelvec[] = {1.0,1.0,1.0};
float redvec[] = {1.0,0.0,0.0};

long xo,yo,xsize,ysize;
long lgld,lgld1;
int menu,menu1;
int ind5;
short mx,my;

int mode,job,exitmode;
char filename[30],sruninfo[50],addinfo[50],showinfo[50];
char surfaceinfo[50],intsinfo[50],sgeomview[50];
int max_size;

main(argc,argv)
int argc;
char *argv[];
{
    char *stroka[100],name[100];
    int i;

    FILE *fp1,*fp2,*fopen();
    if (argc< 3)
        printf("usage: geosphere [name of file] (number of points after interpolation)\n");
    else {
        sprintf(sruninfo,"%s",PATH,"srun.info");
        sprintf(addinfo,"%s",PATH,"adds.info");
        sprintf(showinfo,"%s",PATH,"show.info");
        sprintf(intsinfo,"%s",PATH,"ints.info");
    }

    printf(surfaceinfo,"%s",PATH,"surface.info");
    printf(sgeomview,"%s",PATH,"sgeomview",PLACE),
    max_size = atoi(argv[2]);
    strcpy(filename,argv[1]);
    initialise();
    concave(TRUE);
    Drawobj();

    while('exitmode'){
        while(!qtest()){
            Drawobj();
            swapbuffers();
            processinput();
        }
        winclose(lgld);
    }

    initialise()
    {
        job = -1;
        exitmode = 0;
        keepaspect(1,1);
        foreground(0);
        psize(700,700);
        lgld = winopen("geosphere");
        RGBmode();
        doublebuffer();
        gconf(lgld);
        ortho(0.0,1.0,0.0,1.0,-1.0,1.0);
        c3f(backvec1);
        clear();
        qdevice(RIGHTMOUSE);
        menu = defpup("GEOSPHERE %t INFO IGO ");
        menu1 = defpup("EXIT GEOSPHERE %t YES I NO ");
        tie(RIGHTMOUSE,MOUSEX,MOUSEY);
        getorigin(&xo,&yo);
        getsize(&xsize,&ysize);
    }

    Drawobj()
    {
        ffonthandle f;
        ffonthandle fsize;

        int i,k,ind,angle;
        float xo,yo,xre,yre,size,rad,rad1;
        char *str[15],str1[100],str11[100],str12[100];
        char string[50],st[100];
        str[0] = "latlon";
        str[1] = "triangu-";
        str[2] = "geosphere";
        str[3] = "run";
        str[4] = "latlon";
        str[5] = "interpo-";
        str[6] = "change";
        str[7] = "data ";
        str[8] = "the model";
        str[9] = "view ";

        size = 18.0;
        rad = 0.1;

```


geoform2.c

```

radl = 0.09;
sprintf(str1, "%s %d ", filename, max_size);
printf(str1, " GEOSPHERE ");

mainit();
{
    f = fndfont("Times-Roman");
    fsize = fmscalefont(f, size);
    fsetfont(fsize);

    c3f(backvec);
    rectf(0.0, 0.0, 1.0, 1.0);
    c3f(whitevec2);
    rectf(0.3, 0.05, 0.7, 0.15);
    cmov2(0.35, 0.095);
    c3f(bluevec);
    fprstr((unsigned char *) str1);
    rectf(0.3, 0.85, 0.7, 0.95);
    c3f(yelvec);
    cmov2(0.34, 0.89);
    fprstr((unsigned char *) str1);

    x0 = 0.1; y0 = 0.2;
    ind = 0;
    for(i = 0; i < 3; i++) {
        for(k = 0; k < 2; k++) {
            if(i * k != 1) {
                xre = x0 + i * 0.3 + rad;
                yre = y0 + k * 0.4 + rad;
                c3f(whitevec2);
                rectf(xre - rad, yre - rad, xre + rad, yre + rad);
                if(ind != job)
                    c3f(bluevec);
                else c3f(redvec);
                rectf(xre - radl, yre - radl, xre + radl, yre + radl);
                ind++;
            }
        }
        ind = 0;
        for(i = 0; i < 3; i++) {
            for(k = 0; k < 2; k++) {
                if(i * k != 1) {
                    xre = x0 + i * 0.3;
                    yre = y0 + k * 0.4;
                    c3f(yelvec);
                    cmov2(xre + 0.02, yre + 0.07);
                    fprstr((unsigned char *) str(ind));
                    cmov2(xre + 0.03, yre + 0.13);
                    fprstr((unsigned char *) str(ind + 1));
                    ind = ind + 2;
                }
            }
        }
        icture();

        rocessinput();

        ort val;
        at dev, l, x, m;

```

```

long menuval;
l = k = m = 0;

while(qtest()) {
    Drawobj();
    switch(qread(&val)) {
        case (REDRAW):
            reshapeviewport();
            getorigin(&x0, &y0);
            getsize(&xsize, &ysize);
            while(!qtest()) {
                Drawobj();
                swapbuffers();
            }
            break;
        case (RIGHTMOUSE):
            if(val) {
                qread(&mx);
                qread(&my);
                k++;
                if(k == 1) {
                    choice();
                    if(job == -2) {
                        menuval = dopup(menu);
                        switch(menuval) {
                            case 1:
                                exitmode = 1;
                                break;
                            case 2:
                                exitmode = 0;
                                break;
                            default:
                                break;
                        }
                    }
                }
            }
        if(job > -1) {
            menuval = dopup(menu);
            switch(menuval) {
                case INFO:
                    mode = INFO;
                    Drawobj();
                    swapbuffers();
                    job = -1;
                    choice();
                    break;
                case GO:
                    mode = GO;
                    Drawobj();
                    swapbuffers();
                    choice();
                    job = -1;
                    break;
            }
        }
        default:
            job = -1;
            break;
    }
}

```

```

    }
    if(mode == GO){
        ints_(&ind5,&ind7);
        system("rm saddual stempor");
        job = -1;
    }
    if (mode == INFO){
        try(intsinfo);
        job = -1;
    }
    }
    }
    }

    else if ((xco<0.9)&&(xco>0.7))
    if (yco>0.2&&yco<0.4) {
        if (mode == GO) {
            job = -1;
            adds_(filename,&max_size,&ind5,&ind7);
        }
        if (mode == INFO) {
            try(showinfo);
            job = -1;
        }
    }
    else if (yco>0.6&&yco<0.8)
    {
        if (mode == GO) {
            system("sgeomview<triang &");
            sleep(10);
            job = -1;
        }
        if (mode == INFO) {
            try(surfaceinfo);
            job = -1;
        }
    }
    }

    choicel()
    {
        float xco,yco;
        xco = (float)(mx -xo)/(float)xsize;
        yco = (float)(my -yo)/(float)ysize;
        if ((xco<0.3)&&(xco>0.1)) {
            if (yco<0.4&&yco>0.2)
            {
                if (mode == GO)
                {
                    adds_(filename,&max_size,&ind5,&ind6);
                    job = -1;
                }
                if (mode == INFO)
                {
                    try(addsinfo);
                    job = -1;
                }
            }
            else if (yco>0.6&&yco<0.8)
            {
                if (mode == GO)
                {
                    sleep(2);
                    job = 0;
                    Drawobj();swapbuffers();
                    adds_(filename,&max_size,&ind5,&ind6);
                    if (ind5 == 0) {
                        job = 2;
                    }
                    Drawobj();swapbuffers();
                    ints_(&ind5,&ind7);
                }
                else
                {
                    job = 2;
                    Drawobj(); swapbuffers();
                    ints_ (&ind5,&ind6);
                }
                job = 4;
                Drawobj(); swapbuffers();
                system("rm saddual stempor");
                sleep(10);
                job = -1;
            }
            if (mode == INFO)
            {
                try(sruninfo);
                job = -1;
            }
        }
        else if ((xco<0.6)&&(xco>0.4)) {
            if (yco<0.4&&yco>0.2)
            {
                job = 2;
            }
            else if (xco<0.9&&(xco>0.7)) {
                if (yco<0.4&&yco>0.2)
                {
                    job = 3;
                }
            }
        }
    }
}

```

```

else if (yco>0.664yco<0.8)
{
    job = 4;
}

if (xco<0.7 && xco>0.3)
{
    if (yco>0.85 && yco<0.95)
    {
        job = -2;
    }
}

picture()
{
    concave(TRUE);
    c3f(bluevec);
    bgnpolygon();
    v2f(&compas[8][0]);
    v2f(&compas[1][0]);
    v2f(&compas[9][0]);
    v2f(&compas[5][0]);
    v2f(&compas[10][0]);
    v2f(&compas[3][0]);
    v2f(&compas[11][0]);
    v2f(&compas[7][0]);
    v2f(&compas[8][0]);
    endpolygon();
    c3f(yelvec);
    bgnclosedline();
    v2f(&compas[8][0]);
    v2f(&compas[1][0]);
    v2f(&compas[9][0]);
    v2f(&compas[5][0]);
    v2f(&compas[10][0]);
    v2f(&compas[3][0]);
    v2f(&compas[11][0]);
    v2f(&compas[7][0]);
    v2f(&compas[8][0]);
    endclosedline();
    move2(compas[9][0],compas[9][1]);
    draw2(compas[11][0],compas[11][1]);
    move2(compas[8][0],compas[8][1]);
    draw2(compas[10][0],compas[10][1]);
    c3f(bluevec);
    bgnpolygon();
    v2f(&compas[4][0]);
    v2f(&compas[5][0]);
    v2f(&compas[6][0]);
    v2f(&compas[7][0]);
    v2f(&compas[4][0]);
    endpolygon();
    c3f(yelvec);
    bgnclosedline();
    v2f(&compas[4][0]);
    v2f(&compas[5][0]);
    v2f(&compas[6][0]);
    v2f(&compas[7][0]);
    v2f(&compas[4][0]);
    endclosedline();
    move2(compas[0][0],compas[0][1]);
    draw2(compas[2][0],compas[2][1]);
    move2(compas[4][0],compas[4][1]);
    draw2(compas[1][0],compas[1][1]);
    move2(compas[3][0],compas[3][1]);
    draw2(compas[6][0],compas[6][1]);
}

```

geomv.c

```

#include <stdio.h>
#include <math.h>
#include <gl.h>
#include "panel.h"
#include <device.h>
#include <string.h>
#define maxcol 256
#define k 3.5
#define ANY -1
#define SIZE 50000

float color_palette[maxcol][3];
int ncol;
Object Vobj, sphere, Vobj1;
float wx2, wy2, wz2, scal_par;
typedef float Vec[3];

typedef struct {
    int p_nsides;
    Vec *p;
    int *s;
} Polygon;

int negflag;
float backvec[] = {0.8, 0.6, 0.8}; /*
float backvec[] = {0., 0., 0.};
float blackvec[] = {0.0, 0.0, 0.0};
float redvec[] = {1.0, 0.0, 0.0};
float whitevec[] = {1.0, 1.0, 1.0};
float yelvec[] = {1.0, 1.0, 0.0};

float idmat[] = {1.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0};

float OldMatrix[16];

Polygon *
makePolygon(nsides)
int nsides;
{
    Polygon * p;
    p = (Polygon *) malloc (sizeof(Polygon));
    p -> p_nsides = nsides;
    p -> p = (Vec *) calloc(nsides, sizeof(Vec));
    p -> s = (int *) malloc(3 * sizeof(int));
    return p;
}

Vec
V[SIZE];
Polygon * P[SIZE];
ec N[SIZE], N1[SIZE];
nt npoints, npolys;
ec mins = {1000000, 1000000, 1000000};
ec maxs = {-1000000, -1000000, -1000000};
loat V1[SIZE], V2[SIZE][3];

#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))
float Max, scal;

#define MKSLIDER(v, n, b, t, i) \
    v = pnl_mkact(pnl_slider); \
    v -> label = n; \
    v -> minval = b; \
    v -> maxval = t; \
    v -> val = i; \
    v -> w = 0.6; \
    v -> h = 4.0; \
    v -> x = x; v -> y = y; \
    pnl_addact(v, p)

#define LABEL(v, n) \
    v = pnl_mkact(pnl_label); \
    v -> label = n; \
    v -> x = x; \
    v -> y = y; \
    pnl_addact(v, p)

#define BUTTON(v, n, b, t, i) \
    v = pnl_mkact(pnl_radio_button); \
    v -> labeltype = PNL_LABEL_BOTTOM; \
    v -> label = n; \
    v -> minval = b; \
    v -> maxval = t; \
    v -> val = i; \
    v -> x = x; v -> y = y; \
    v -> h = hl; v -> w = w; \
    pnl_addact(v, p)

#define BUTTON2(v, n, b, t, i) \
    v = pnl_mkact(pnl_toggle_button); \
    v -> labeltype = PNL_LABEL_BOTTOM; \
    v -> label = n; \
    v -> minval = b; \
    v -> maxval = t; \
    v -> val = i; \
    v -> x = x; v -> y = y; \
    v -> h = hl; v -> w = w; \
    pnl_addact(v, p)

#define BUTTON3(v, n, b, t, i) \
    v = pnl_mkact(pnl_button); \
    v -> labeltype = PNL_LABEL_BOTTOM; \
    v -> label = n; \
    v -> minval = b; \
    v -> maxval = t; \
    v -> val = i; \
    v -> x = x; v -> y = y; \
    v -> h = hl; v -> w = w; \
    pnl_addact(v, p)

Actuator * editPalette, * distance, * theta, * phi, * model, * xlight1, * ylight1, * zlight1, * xlight2, * ylight2, * zlight3, * xlight3, * ylight3, * surface, * zsc, * map, * twist, * size, * shlx, * shly, * smooth, * xsc, * ysc, * tumble, * savePosition, * getPosition, * myMouse, * savePict, * inputStr, * box, * reset, * quit, * l1, * l2, * l3, * light1, * light2, * light3;

DefPanel()
{

```

```

Panel * p ;
float x = 0.0, y = 0.0, sep = 0.6, sepl=1.1, sep2 = 0.4;
float hl= 0.5, w = 1.0;

p = pnl_mkpanel() ;
p->label = "Control Panel v.10" ;
p->ppu = 36.0 ;
BUTTON3(quit, "QUIT", 0.0, 1.0, 0.0) ;
x = 4.5; y = quit->h*sep;
BUTTON3(savepict, "SAVE PICTURE", 0.0, 1.0, 0.0) ;
y+= savepict->h*sep;
inputstr = pnl_mkact (pnl_typein);
inputstr->labeltype = PNL_LABEL_TOP_LEFT;
inputstr->label = "save as: ";
inputstr->x = 0.0; inputstr->y = y;
pnl_addact(inputstr, p);
y += inputstr->h*2*sep;
x = 2.0;
BUTTON2(box, "SHOW BOX", 0.0, 1.0, 0.0) ;
x = 5.0;
BUTTON3(reset, "RESET", 0.0, 1.0, 0.0) ;
y += reset->h*2*sep;
x = 1.0;
BUTTON3(savepos, "Save Position", 0.0, 1.0, 0.0) ;
x = 4.5;
BUTTON3(getpos, "Get Position", 0.0, 1.0, 0.0) ;
x = 8.0;
BUTTON3(editpalette, "Edit Palette", 0.0, 1.0, 0.0) ;
x = 0.5; y += w*sep;
LABEL(12, "ROTATE");
x = 3.0;
LABEL(11, "TRANSLATE");
x = 6.0;
LABEL(13, "SCALE");
y+=hl*sep; x = 0.0;
MKHSLIDER(phi, "X", 0.0, 3600.0, 0.0) ;
x+=phi->w+0.3;
MKHSLIDER(twist, "Y", 0.0, 3600.0, 0.0) ;
x+=phi->w+0.3;
MKHSLIDER(theta, "Z", 0.0, 3600.0, 0.0) ;
x+=phi->w+0.6;
MKHSLIDER(shix, "X", -1.0*Max, Max, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(shiy, "Y", -1.0*Max, Max, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(distance, "Z", -2.0, 2.0, 0.0) ;
x+=theta->w+0.6;
MKHSLIDER(xsc, "X", 1.1, 1.0, 1.0) ;
x+=theta->w+0.3;
MKHSLIDER(ysc, "Y", 1.1, 1.0, 1.0) ;
x+=theta->w+0.3;
MKHSLIDER(zsc, "Z", 1.1, 1.0, 1.0) ;
x+=theta->w+0.6;
MKHSLIDER(size, "SIZE", 0.5, 5.0, 2.0/Max) ;
y+=size->h+1.5*sep;
x = 0.5;
BUTTON2(ilight1, "LIGHT", 0.0, 1.0, 1.0) ;
x = 3.5;
BUTTON2(ilight2, "RED LIGHT", 0.0, 1.0, 0.0) ;
x = 6.5;
BUTTON2(ilight3, "BLUE LIGHT", 0.0, 1.0, 0.0) ;
y+=hl*sep;
x = 0.0;
MKHSLIDER(xlight1, "X", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(ylight1, "Y", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(zlight1, "Z", -1.0, 1.0, 1.0) ;
x+=theta->w+0.6;
MKHSLIDER(xlight2, "X", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(ylight2, "Y", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(zlight2, "Z", -1.0, 1.0, 1.0) ;
x+=theta->w+0.6;
MKHSLIDER(xlight3, "X", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(ylight3, "Y", -1.0, 1.0, 0.0) ;
x+=theta->w+0.3;
MKHSLIDER(zlight3, "Z", -1.0, 1.0, 1.0) ;
y+=zlight2->h+1.5*sep;
x = 2.0;
BUTTON (smooth, "Gouraud Lighting", 0.0, 1.0, 0.0) ;
x = 6.0;
BUTTON (map, "3D Map", 0.0, 1.0, 0.0) ;
y +=hl*sep; x = 4.0;
BUTTON2 (tumble, "Tumble", 0.0, 1.0, 0.0) ;
y+=hl*sep; x = 2.0;
BUTTON(surf, "Lighting", 0.0, 1.0, 0.0) ;
x = 6.0;
BUTTON(model, "Wireframe", 0.0, 1.0, 0.0) ;
myMouse = pnl_mkact(pnl_mouse);
pnl_addact(myMouse, p);

float shiftx, shifty, tempo ;

ReadObject()
{
    int i, j, ku, nverts, index[3], nsides ;
    Vec tmp1, tmp2 ;
    scanf("%d %d\n", & npolys, & npoints) ;

    fprintf(stderr, "%d points, %d polys\n", npoints, npolys) ;
    /*
        for (j = 0 ; j < 3 ; j++) {
            mins[j] = 1e10;
            maxs[j] = -1e10;
        }
        for (i = 0 ; i < npoints ; i++) {
            scanf("%f %f %f\n", &(V[i][0]), &(V[i][1]), &(V[i][2])) ;
        }
    */
    /*Lala: I changed here */
    for(i = 0; i < npoints; i++)
        for (ku = 0 ; ku < 3 ; ku++) {
            mins[ku] = min(mins[ku], V[i][ku]) ;
            maxs[ku] = max(maxs[ku], V[i][ku]) ;
        }
    Max = max(maxs[0]-mins[0], maxs[1]-mins[1]) ;
    Max = max(Max, maxs[2]-mins[2]) ;
    shiftx = 0.5*(maxs[0]+mins[0]) ;
    shifty = 0.5*(maxs[1]+mins[1]) ;
}

```

```

for (i = 0; i < npoints; i++) {
    tempo = V[i][1];
    V[i][0] = V[i][0] - shiftx;
    V[i][1] = tempo - shifty;
}

printf("shifts %f %f\n", shiftx, shifty);

for (i = 0; i < npolys; i++) {
    P[i] = MakePolygon(3);
    for (j = 0; j < 3; j++) {
        scanf("%d", &index[j]);
        P[i] -> s[j] = index[j] - 1;
        P[i] -> p[j] = V + (index[j] - 1);
        V2[i][j] = V[index[j] - 1][2];
        for (ku = 0; ku < 3; ku++) {
            mins[ku] = min(mins[ku], V[index[j] - 1][ku]);
            maxs[ku] = max(maxs[ku], V[index[j] - 1][ku]);
        }
    }
    V1[i] = V2[i][0] + V2[i][1] + V2[i][2];
    V1[i] = V1[i]/3.;

    printf(stderr, "bbx: %f - %f\n", mins[0], maxs[0]);
    printf(stderr, "bbx: %f - %f\n", mins[1], maxs[1]);
    printf(stderr, "bbx: %f - %f\n", mins[2], maxs[2]);
    /*
    *mins[2] = 0.;
    */
    scal = ncol / (maxs[2] - mins[2]);

    Max = max(maxs[0] - mins[0], maxs[1] - mins[1]);
    Max = max(Max, maxs[2] - mins[2]);
    shiftx = 0.5 * (maxs[0] + mins[0]);
    shifty = 0.5 * (maxs[1] + mins[1]);
    scal_par = Max/2.0;
    printf("shifts %f %f\n", shiftx, shifty);

    for (i = 0; i < npolys; i++) {
        VecSub(P[i] -> p[0], P[i] -> p[1], tmp1);
        VecSub(P[i] -> p[1], P[i] -> p[2], tmp2);
        VecCross(tmp1, tmp2, N[i]);
        VecNormalize(N[i]);
        if (negflag) {
            if (N[i][0] < 0.) N[i][0] = -N[i][0];
            if (N[i][1] < 0.) N[i][1] = -N[i][1];
            if (N[i][2] < 0.) N[i][2] = -N[i][2];
        }
    }
}

else {
    N[i][0] = -N[i][0];
    N[i][1] = -N[i][1];
    N[i][2] = -N[i][2];
}

for (i = 0; i < npolys; i++) {
    N[i][0] = 0.0;
    N[i][1] = 0.0;
    N[i][2] = 0.0;
}

N[i][2] = 0.0;
}

for (i = 0; i < npolys; i++) {
    for (j = 0; j < 3; j++) {
        N[i] -> s[j] -> p[j] -> N[i][0];
        N[i] -> s[j] -> p[j] -> N[i][1];
        N[i] -> s[j] -> p[j] -> N[i][2];
    }
    for (i = 0; i < npolys; i++) {
        for (j = 0; j < 3; j++) {
            N[i] -> s[j] -> p[j] -> N[i][0];
            N[i] -> s[j] -> p[j] -> N[i][1];
            N[i] -> s[j] -> p[j] -> N[i][2];
        }
    }
    VecNegate(a);
    Vec a;
    {
        a[0] = 0 - a[0];
        a[1] = 0 - a[1];
        a[2] = 0 - a[2];
    }
    VecSub(a, b, c);
    Vec a, b, c;
    {
        c[0] = a[0] - b[0];
        c[1] = a[1] - b[1];
        c[2] = a[2] - b[2];
    }
    VecCross(a, b, c);
    Vec a, b, c;
    {
        c[0] = a[1] * b[2] - b[1] * a[2];
        c[1] = - (a[0] * b[2] - b[0] * a[2]);
        c[2] = a[0] * b[1] - b[0] * a[1];
    }
    VecNormalize(a);
    Vec a;
    {
        float l;
        l = (float) sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]);
        a[0] /= l;
        a[1] /= l;
        a[2] /= l;
    }
    float shinnymaterial[] = { SPECULAR, 1.0, 0.2, 0.8,
        EMISSION, 0.8, 0.8, 0.0,
        DIFFUSE, 0.8, 0.0, 0.0,
        AMBIENT, 0.2, 0.0, 0.0,
        SHININESS, 3.0,
        LMNULL };
    float sun[] = { LCOLOR, 1.0, 1.0, 1.0,
        POSITION, 0.0, 0.0, 1.0, 0.0,
        LMNULL };
}

```

geomv.c

4

```

float sun2[] = { LCOLOR,1.0, 0 0,0.0,
                POSITION, 0.0, 0.0, 1.0,0.0,
                LNULL } ;

float sun3[] = { LCOLOR,0.0, 0.0,1.0,
                POSITION, 0.0, 0.0, 1.0,0.0,
                LNULL } ;

int tum , tumble;

DrawObject()
{
    Mouse * mymouse;
    short xmouse,ymouse;
    char * str[100];
    int i, j,col_ind ;
    int temp0,temp1,temp2,temp3,tumble,editPal,editPos,savePos;
    float sit,shx,shy,x_sc,y_sc,z_sc;
    temp1=model -> val;
    temp2=surface -> val;
    temp0=smooth -> val;
    temp3=map -> val;
    siz = size -> val;
    shx = shx -> val;
    shy = shy -> val;
    x_sc = xsc->val;
    y_sc = ysc->val;
    z_sc = zsc->val;
    editPal = editPalette -> val;
    getPos = getPosition -> val;
    savePos = savePosition -> val;

    xmouse = PNL_ACCESS(Mouse, myMouse, x);
    ymouse = PNL_ACCESS(Mouse, myMouse, y);
    /*get palette()*/get_surface();
    sprintf(str,"%s %d %s",PATH,"mycedit",ncol,"g");
    if(editPal == 1.0) system(str);
    if(savePos == 1.0) save_position();
    if(getPos == 1.0) get_position();
    if(savePic->val) save_picture();
    if(reset->val) make_reset();
    reshapeviewport();
    c3f(backvec);
    clear();
    zclear();
    MakeVobj();
    CallObj(Vobj);
    /*MakeLight(xmouse,ymouse);*/
    MakeLight1();
    /*
    callObj(Vobj);*/
    if(box->val) Drawbox();
    /*translate(-1.0*shiftx,-1.0*shifty,0.0);*/
    ChangeStack();
    if (temp1 == 1) {
        .mbind(MATERIAL, 0);
        .mbind(LIGHT0, 0);
        .mbind(LIGHT1, 0);
        .mbind(LIGHT2, 0);
        .mbind(LMODEL, 0);
        for (i = 0; i < npolys; i++) {
            c3f(redvec);
            /*c3f(backvec)*/
            bgnclosedline();
            for (j = 0; j < P[i] -> p_nsides; j++) {
                v3f(P[i]->p[j]);
            }
            endclosedline();
        }
        if (temp3 == 1) {
            .mbind(MATERIAL, 0);
            .mbind(LIGHT0, 0);
            .mbind(LIGHT1, 0);
            .mbind(LIGHT2, 0);
            .mbind(LMODEL, 0);
            for (i = 0; i < npolys; i++) {
                col_ind=(int) ((V1[i]-mins[2])*scal);
                col_ind=min(ncol-1,max(0,col_ind));
                c3f(&color_palette[ col_ind[0]]);
            }
            bgnpolygon();
            for (j = 0; j < P[i] -> p_nsides; j++) {
                col_ind=(int) ((V2[i][j]-mins[2])*scal);
                col_ind=min(ncol-1,max(0,col_ind));
                c3f(&color_palette[ col_ind[0]]);
            }
            endpolygon();
        }
        if(temp2 || temp0) {
            .mbind(MATERIAL, 1);
            if(light1->val) .mbind(LIGHT0, 1); else .mbind(LIGHT0, 0);
            if(light2->val) .mbind(LIGHT1, 2); else .mbind(LIGHT1, 0);
            if(light3->val) .mbind(LIGHT2, 3); else .mbind(LIGHT2, 0);
            .mbind(LMODEL, 1);
            for (i = 0; i < npolys; i++) {
                bgnpolygon();
                for (j = 0; j < P[i] -> p_nsides; j++) {
                    c3f(yelvec);
                    if(temp2) n3f(N[i]);
                    if(temp0) n3f(N1[P[i]->s[j]]);
                    v3f(P[i]->p[j]);
                }
                endpolygon();
            }
            swapbuffers();
            make_reset();
        }
        def_simple_light_calc()
        {
            .lndef(DEFMATERIAL, 1, 19, shlnymaterial);
            .lndef(DEFLIGHT, 1, 10, sun1);
            .lndef(DEFMODEL, 1, 0, NULL);
        }
        use_simple_light_calc()
        {
            .mbind(MATERIAL, 1);
            .mbind(LIGHT0, 1);
            .mbind(LMODEL, 1);
        }
    }
}

```

```

int exitmode;

DO_JOB ()
{
    int mmm;
    mmm=80;
    c3f(backvec) ;
    clear() ;
    zclear() ;
    while (!exitmode) {
        pnl_dopanel() ;
        reshapeviewport() ;
        DrawObject();
        tumble = tumble -> val;
        if(tumble == 1)
            tum += mmm ;
        exitmode = quilt->val;
    }
}

float kvadrat1[] = {-0.5,-0.5,-0.5};
float kvadrat2[] = {-0.5,0.5,-0.5};
float kvadrat3[] = {0.5,0.5,-0.5};
float kvadrat4[] = {0.5,-0.5,-0.5};

Drawbox ()
{
    char string1[5],string2[5],string3[5],string4[5];

    sprintf(string1,"%s","X");
    sprintf(string2,"%s","Y");
    sprintf(string3,"%s","Z");
    sprintf(string4,"%s","O");
    c3f(whitevec);

    cmov(0.50,-0.47,-0.50);
    charstr(string1);
    cmov(-0.50,0.53,-0.50);
    charstr(string2);
    cmov(-0.53,-0.53,0.53);
    charstr(string3);
    cmov(-0.47,-0.47,-0.47);
    charstr(string4);

    pushmatrix();
    bgnclosedline();
    v3f(kvadrat1);
    v3f(kvadrat2);
    v3f(kvadrat3);
    v3f(kvadrat4);
    endclosedline();
    popmatrix();
    pushmatrix();
    translate(0.0,0.0,1.0);
    bgnclosedline();
    v3f(kvadrat1);
    v3f(kvadrat2);
    v3f(kvadrat3);
    v3f(kvadrat4);
    endclosedline();
    popmatrix();
    rotate(-90.0,'y');
    bgnclosedline();
    v3f(kvadrat1);
    v3f(kvadrat2);
    v3f(kvadrat3);
    v3f(kvadrat4);
    endclosedline();
    popmatrix();
    pushmatrix();
    translate(-1.0,0.0,0.0);
    rotate(-90.0,'y');
    bgnclosedline();
    v3f(kvadrat1);
    v3f(kvadrat2);
    v3f(kvadrat3);
    v3f(kvadrat4);
    endclosedline();
    popmatrix();
    pushmatrix();
    rotate(-90.0,'y');
    bgnclosedline();
    v3f(kvadrat1);
    v3f(kvadrat2);
    v3f(kvadrat3);
    v3f(kvadrat4);
    endclosedline();
    popmatrix();
}

init_palette()
{
    int i,j;
    float dx1;
    for(i = 0;i<3;i++)
        for(j = 0;j<ncol;j++)
            color_palette[j][i] = 0.0;

    for(i = 0;i<3;i++) {
        dx1 = 4./ncol;
        if(i == 0) {
            for(j = ncol;j>= (int) (3*ncol/4)+1;j--) {
                color_palette[j][i] = 1.0;
            }
            for(j = (int) (3*ncol/4);j>= ncol/2;j--) {
                color_palette[j][i] = dx1*(j - (int) (ncol/2));
            }
        }
        if(i == 1) {
            for(j = (int) (ncol);j>= (int) (3*ncol/4);j--) {
                color_palette[j][i] = 1 - dx1*(j - (int) (3*ncol/4));
            }
            for(j = (int) (3*ncol/4);j>= (int) (ncol/4);j--) {
                color_palette[j][i] = 1.0;
            }
            for(j = (int) (ncol/4);j>= 0;j--) {
                color_palette[j][i] = dx1*j;
            }
        }
        if(i == 2) {
            for(j = ncol/2;j>= (int) (ncol/4);j--) {
                color_palette[j][i] = 1.0 - dx1*(j-ncol/4);
            }
            /* printf("%f %d %d \n", color_palette[j][i],1,j); */
        }
    }
}

```



```

for(j = (int)(ncol/4); j>=0; j--) {
    color_palette[j][1] = 1.0;
}

)
)
)

get_palette()
{
    int j;
    FILE *fp, *fopen();
    fp = fopen("color_palette", "r");

    if(fp != NULL) {
        fscanf(fp, "%d", &ncol);
        for(j = 0; j<ncol; j++)
            fscanf(fp, "%f %f", &color_palette[j][0], &color_palette[j][1],
                &color_palette[j][2]);
    }

    fclose(fp);
    scal=ncol/(maxs[2]-mins[2]);

    else {ncol = maxcol; init_palette();}

    pnl_drawpanel();

}

get_surface()
{
    FILE *fp, *fopen();

    p = fopen("lighting", "r");
    if(fp != NULL) {
        scanf(fp, "%f %f %f", &shinymaterial[1], &shinymaterial[2], &shinymaterial[3]);
        scanf(fp, "%f %f %f", &shinymaterial[5], &shinymaterial[6], &shinymaterial[7]);
        scanf(fp, "%f %f %f", &shinymaterial[9], &shinymaterial[10], &shinymaterial[11]);
        scanf(fp, "%f %f %f", &shinymaterial[13], &shinymaterial[14], &shinymaterial[15]);
        scanf(fp, "%f %f %f", &ssunl[1], &ssunl[2], &ssunl[3]);
        #def(DEFMATERIAL, 1, 19, shinymaterial);
        #bind(MATERIAL, 1);
        #def(DEFLIGHT, 10, sunl);
        #bind(LIGHT0, 1);
    }

    close(fp);
    nl_drawpanel();

}

ave_position()
{
    float siz, shx, shy, x_sc, y_sc, z_sc;
    float dis, twl, the, phi;
    FILE *fp, *fopen();
    siz = size->val;
    shx = shx->val;
    shy = shy->val;
    x_sc = xsc->val;
    y_sc = ysc->val;
    z_sc = zsc->val;

    makeobj(Vobj1 = genobj());
    loadmatrix(1dmat);
    ortho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
    /*perspective(fovy, 1.0, 0.1, k*x* Max+(1./ang)) ;*/
    closeobj();

    makeobj(Vobj = genobj());
}

dis = distance ->val;
twl = twist -> val;
the = theta -> val;
ph = phi ->val;

fp = fopen("pnl_position", "w");
fprintf(fp, "%f %f %f %f %f\n", siz, shx, shy, x_sc, y_sc, z_sc);
fprintf(fp, "%f %f %f %f %f\n", dis, twl, the, phi);
fclose(fp);
pnl_drawpanel();

}

get_position()
{
    int temp0, templ, temp2, temp3;
    float siz, shx, shy, x_sc, y_sc, z_sc;
    float dis, twl, the, phi;
    FILE *fp, *fopen();
    fp = fopen("pnl_position", "r");
    if(fp != NULL) {fscanf(fp, "%f %f %f %f %f %f\n", &siz, &shx, &shy, &x_sc, &y_sc, &z_sc);
        fscanf(fp, "%f %f %f %f %f %f\n", &dis, &twl, &the, &phi);
        fclose(fp);
    }

    siz->val = siz;
    shx->val = shx;
    shy->val = shy;
    xsc->val = x_sc;
    ysc->val = y_sc;
    zsc->val = z_sc;
    distance->val = dis;
    twist->val = twl;
    theta->val = the;
    phi->val = phi;
}

pnl_drawpanel()
{
    MakeVobj()
    {
        float siz, shx, shy, x_sc, y_sc, z_sc;
        float x, y, z;
        int thetal;

        siz = size->val;
        shx = shx->val;
        shy = shy->val;
        x_sc = xsc->val;
        y_sc = ysc->val;
        z_sc = zsc->val;

        makeobj(Vobj1 = genobj());
        loadmatrix(1dmat);
        ortho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
        /*perspective(fovy, 1.0, 0.1, k*x* Max+(1./ang)) ;*/
        closeobj();

        makeobj(Vobj = genobj());
    }
}

```

```

loadmatrix(ldmat);
/*perspective(fovy, 1.0, 0.1, k*k*Max*(1./ang));
polarview(distance->val, 0.0, 0.0, 0.0);*/
ortho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
rotate((short)twist -> val, 'y');
rotate((short)theta -> val, 'z');
rotate((short)phi -> val, 'x');
scale(x_sc, y_sc, z_sc);
rotate(tum, 'y');
translate(shv, shv, distance->val);
scale(size->val, size->val, size->val);
closeobj();

makeobj(sphere = genobj());
for(theta = 0; theta < 360; theta = theta + 300) {
    rotate(300, 'y');
    circ(0.0, 0.0, 1.0);
}
closeobj();
}

makeLight(xmouse, ymouse)
ScreenCoord xmouse, ymouse;
{
    long xor, yor;
    float wx1, wy1, wz1;
    getorigin(&xor, &yor);
    xmouse = xmouse - xor;
    ymouse = ymouse - yor;
    mapw(Vob), xmouse, ymouse, &wx1, &wy1, &wz1, &wx2, &wy2, &wz2;
    sun1[5] = wx2; sun1[6] = wy2; sun1[7] = wz2;
    imdef(DEFLIGHT, 1, 10, sun1);
    imbind(LIGHT0, 1);
    pushmatrix();
    translate(wx2, wy2, wz2);
    scale(0.01, 0.01, 0.01);
    c3f(4, sun1[1]);
    callobj(sphere);
    popmatrix();
}

MakeLight1()
{
    sun1[5] = xlight1->val;
    sun1[6] = ylight1->val;
    sun1[7] = zlight1->val;
    imdef(DEFLIGHT, 1, 10, sun1);
    imbind(LIGHT0, 1);
    sun2[5] = xlight2->val;
    sun2[6] = ylight2->val;
    sun2[7] = zlight2->val;
    imdef(DEFLIGHT, 2, 10, sun2);
    imbind(LIGHT1, 2);
    sun3[5] = xlight3->val;
    sun3[6] = ylight3->val;
    sun3[7] = zlight3->val;
    imdef(DEFLIGHT, 3, 10, sun3);
    imbind(LIGHT2, 3);
}

save_picture()
{
    char savePict[50], *out_rgb, outStr[20];
    long xor, yor, xsize, ysize;
    getorigin(&xor, &yor);
    getsize(&xsize, &ysize);
    out_rgb = PNL_ACCESS(TYPEIN, inputStr, str);
    sprintf(savePict, "%s %d %d %d", "savesave ", out_rgb, xor,
        xor+xsize, yor, yor+ysize);
    system(savePict);
}

#define RESET(v) v->val = v->initval

make_reset()
{
    RESET(distance);
    RESET(theta);
    RESET(phi);
    RESET(twist);
    RESET(size);
    RESET(shx);
    RESET(shy);
    RESET(xsc);
    RESET(yse);
    RESET(zsc);
    RESET(tumble);
}

pnl_drawpanel()
{
    make_reset();
    loadmatrix(ldmat);
    getmatrix(OldMatrix);
}

ChangeStack()
{
    loadmatrix(OldMatrix);
    if(twist->active) rotate((short)twist -> val, 'y');
    if(theta->active) rotate((short)theta -> val, 'z');
    if(phi->active) rotate((short)phi -> val, 'x');
    if(xsc->active || yse->active || zsc->active) scale(xsc->val, yse->val, zsc->val);
    if(size->active) scale(size->val, size->val, size->val);
    getmatrix(OldMatrix);
}

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fp, *fopen();
    int i, j, temp2;
    short val;
    int dev;
    ncol = maxcol;
    negflag = 0;
}

```

```
if (argc == 2)
    negflag = 1;
printf("%d negflag\n", negflag);
init_palette();
ReadObject();

keepaspect(1, 1);
foreground();
winopen("surface");
RGBmode();
doublebuffer();
gconfig();

lsetdepth(0, 0x7FFFFFFF);
zbuffer(TRUE);

mmode(MVIEWING);

DefPanel();
pnl_needredraw();
def_simple_light_calc();
loadmatrix(idmat);
scale(size->val, size->val, size->val);
/*translate(-1.0*shiftx, -1.0*shifty, 0.0);*/
getmatrix(oldMatrix);

DrawObject();
add_event(winget(), REDRAW, ANY, DO_JOB(), 0);
qdevice (REDRAW);
}
```

geomview.c

```

#include <stdio.h>
#include <math.h>
#include <gl.h>
#include "panel.h"
#include <device.h>
#include <string.h>
#include "cpath.h"
#define maxcol 256
#define MATRIXSIZE 16
#define LIGHTINDEX 10
#define k 3.5
#define ANY -1
#define SIZE 500000

float color_palette[maxcol][3];
int ncol;
Object Vobj, sphere, Vobj1;
float wx2, wy2, wz2, scal_p;
typedef float Vec[3];
typedef struct {
    int P_nsides;
    Vec *p;
    int *s;
} Polygon;

int negflag;
float backvec[] = {0.8, 0.8, 0.8};
float blackvec[] = {0., 0., 0.1};
float redvec[] = {1.0, 0.0, 0.0};
float whitevec[] = {1.0, 1.0, 1.0};
float yelvec[] = {1.0, 1.0, 0.0};

Matrix ldmatrix = {1.0, 0.0, 0.0, 0.0,
                   0.0, 1.0, 0.0, 0.0,
                   0.0, 0.0, 1.0, 0.0,
                   0.0, 0.0, 0.0, 1.0};
Matrix OldMatrix, NewMatrix;
Polygon *
makePolygon(nsides)
int nsides;
{
    Polygon * p;
    p = (Polygon *) malloc (sizeof(Polygon));
    p->P_nsides = nsides;
    p->p = (Vec *) calloc(nsides, sizeof(Vec));
    p->s = (int *) malloc(3* sizeof(int));
    return p;
}

Vec V[SIZE];
Polygon * P[SIZE];
int N[SIZE], N1[SIZE];
int npoints, npolys;
Vec mins = {1000000, 1000000, 1000000};
Vec maxs = {-1000000, -1000000, -1000000};

float V1[SIZE], V2[SIZE][3];
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))
float Max, scal;
#define MKHSLIDER(v, n, b, t, i) \
    v = pnl_mkact(pnl_vslider); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->w = 0.6; \
    v->h = 4.0; \
    v->x = x; v->y = y; \
    pnl_addact(v, p)
#define LABEL(v, n) \
    v = pnl_mkact(pnl_label); \
    v->label = n; \
    v->x = x; \
    v->y = y; \
    pnl_addact(v, p)
#define BUTTON(v, n, b, t, i) \
    v = pnl_mkact(pnl_radio_button); \
    v->labeltype = PNL_LABEL_BOTTOM; \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hi; v->w = w; \
    pnl_addact(v, p)
#define BUTTON2(v, n, b, t, i) \
    v = pnl_mkact(pnl_toggle_button); \
    v->labeltype = PNL_LABEL_BOTTOM; \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hi; v->w = w; \
    pnl_addact(v, p)
#define BUTTON3(v, n, b, t, i) \
    v = pnl_mkact(pnl_button); \
    v->labeltype = PNL_LABEL_BOTTOM; \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hi; v->w = w; \
    pnl_addact(v, p)

Actuator * editPalette, * distance, * theta, * phi, * model, * xlight1, * ylight1,
* zlight1, * xlight2, * ylight2, * xlight3, * ylight3, * zlight3, * surface,
* zsc, * map, * twist, * size, * shix, * shiy, * smooth, * xsc, * ysc, * reslight, * savePosition,
* getposition, * myMouse, * savePic, * inputStr, * box, * reset, * quit, * ll, * l2, * l3,
* light1, * light2, * light3;

```

gcomview.c

2

```

DefPanel()
{
    Panel * p ;
    float x = 0.0, y = 0.0, sep = 0.6, sep1=1.1, sep2 = 0.4;
    float hl= 0.5, w = 1.0;

    p = pnl_mkpanel();
    p->label = "Control Panel v.10" ;
    p->ppu = 36.0 ;
    BUTTON3(quit, "QUIT", 0.0, 1.0, 0.0);
    x = 4.5; y = quit->h*sep;
    BUTTON3(savePic, "SAVE PICTURE", 0.0, 1.0, 0.0);
    y+= savePic->h*sep;
    inputStr = pnl_mkact(pnl_typeln);
    inputStr->labelType = PNL_LABEL_TOP_LEFT;
    inputStr->label = "Save as: ";
    inputStr->x = 0.0; inputStr->y = y;
    pnl_addact(inputStr, p);
    y += inputStr->h*2*sep;
    x = 2.0;
    BUTTON2(box, "SHOW BOX", 0.0, 1.0, 0.0);
    x = 5.0;
    BUTTON3(reset, "RESET", 0.0, 1.0, 0.0);
    y += reset->h*2*sep;
    x = 1.0;
    BUTTON3(savePosition, "Save Position", 0.0, 1.0, 0.0);
    x = 4.5;
    BUTTON3(getPosition, "Get Position", 0.0, 1.0, 0.0);
    x = 8.0;
    BUTTON3(editPalette, "Edit Palette", 0.0, 1.0, 0.0);
    x = 0.5; y+= w*sep;
    LABEL(l1, "ROTATE");
    x = 3.0;
    LABEL(l1, "TRANSLATE");
    x = 6.0;
    LABEL(l3, "SCALE");
    y+=hl*sep; x = 0.0;
    MKHSLIDER(phi, "X", -1800.0, 1800.0, 0.0);
    x+=phi->w*0.3;
    MKHSLIDER(twist, "Y", -1800.0, 1800.0, 0.0);
    x+=phi->w*0.3;
    MKHSLIDER(theta, "Z", -1800.0, 1800.0, 0.0);
    x+=phi->w*0.6;
    MKHSLIDER(shiftx, "X", -1.0*Max, Max, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(shly, "Y", -1.0*Max, Max, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(distance, "Z", -2.0, 2.0, 0.0);
    x+=theta->w*0.6;
    MKHSLIDER(xsc, "X", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(ysc, "Y", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(zsc, "Z", -1.0, 1.0, 0.0);
    x+=theta->w*0.6;
    MKHSLIDER(size, "SIZE", -1.0, 1.0, 0.0);
    y+=size->h + 1.5*sep;
    x = 0.5;
    BUTTON2(light1, "LIGHT", 0.0, 1.0, 1.0);
    x = 3.5;
    BUTTON2(light2, "RED LIGHT", 0.0, 1.0, 0.0);
    x = 6.5;
    BUTTON2(light3, "BLUE LIGHT", 0.0, 1.0, 0.0);
    y+=hl*sep;

    x = 0.0;
    MKHSLIDER(xlight1, "X", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(ylight1, "Y", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(zlight1, "Z", -1.0, 1.0, 1.0);
    x+=theta->w*0.6;
    MKHSLIDER(xlight2, "X", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(ylight2, "Y", -1.0, 1.0, 0.0);
    x+=theta->w*0.6;
    MKHSLIDER(zlight2, "Z", -1.0, 1.0, 1.0);
    x+=theta->w*0.3;
    MKHSLIDER(xlight3, "X", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(ylight3, "Y", -1.0, 1.0, 0.0);
    x+=theta->w*0.3;
    MKHSLIDER(zlight3, "Z", -1.0, 1.0, 1.0);
    y+=zlight2->h+1.5*sep;
    x = 4.0;
    BUTTON3 (reslight, "RESET LIGHT", 0.0, 1.0, 0.0);
    y+=hl+1.5*sep; x = 2.0;
    x = 2.0;
    BUTTON (smooth, "Gouraud Lighting", 0.0, 1.0, 0.0);
    x = 6.0;
    BUTTON (map, "3D Map", 0.0, 1.0, 0.0);
    y +=hl+1.5*sep; x = 2.0;
    BUTTON (surface, "Lighting", 0.0, 1.0, 0.0);
    x = 6.0;
    BUTTON (model, "Wireframe", 0.0, 1.0, 0.0);
    myMouse = pnl_mkact(pnl_mouse);
    pnl_addact(myMouse, p);

    float shiftx, shifty, shiftz;

    ReadObject()
    {
        int i, j, k, nverts, indl[3], nsides;
        Vec tmp1, tmp2;
        scanf("%d %d\n", & npolys, & npoints);

        printf(stderr, "%d points, %d polys\n", npoints, npolys);
        /*
        for (j = 0; j < 3; j++) {
            mins[j] = 1e10;
            maxs[j] = -1e10;
        }
        for (i = 0; i < npoints; i++) {
            scanf("%f %f %f\n", & (V[i][0]), & (V[i][1]), & (V[i][2]));
        }
        for (i = 0; i < npolys; i++) {
            P[i] = MakePolygon(3);
            for (j = 0; j < 3; j++) {

```

```

scanf("%d", &indf[j]) ;
l= indf[j]-1;
P[i]-->s[j] = l;
for( ku = 0 ; ku < 3 ; ku++) {
    mins[ku] = min(mins[ku], V[l][ku]) ;
    maxs[ku] = max(maxs[ku], V[l][ku]) ;
}
}

Max = max(maxs[0]-mins[0], maxs[1]-mins[1]) ;
Max = max(Max, maxs[2]-mins[2]) ;
scal=ncol/(maxs[2]-mins[2]) ;
shiftx = 0.5*(maxs[0]+mins[0]);
shifty = 0.5*(maxs[1]+mins[1]);
shiftz = 0.5*(maxs[2]+mins[2]);

for( i = 0; i < npoints ; i++) {
    V[i][0] = V[i][0] - shiftx;
    V[i][1] = V[i][1] - shifty;
    V[i][2] = V[i][2] - shiftz;
}
maxs[?] = maxs[2]-shiftz;
mins[2] = mins[2]-shiftz;

for( i = 0 ; i < npolys ; i++) {
    for( j = 0 ; j < 3 ; j++) {
        l=P[i]-->s[j] ;
        for(ku = 0; ku< 3; ku++)
            (P[i]->p[j])[ku] = V[l][ku];
        v2[i][j] = V[l][2];
    }
    V1[i] = V2[i][0] + V2[i][1] +V2[i][2];
    V1[i] = V1[i]/3.;
}

/*_____*/
scal_par = Max/2.0;

for( i = 0 ; i < npolys ; i++) {
    VecSub(P[i]->p[0], P[i]->p[1], tmp1) ;
    VecSub(P[i]->p[1], P[i]->p[2], tmp2) ;
    VecCross(tmp1, tmp2, N[i]) ;
    VecNormalize(N[i]) ;
    if( negflag )
        if(N[i][0] < 0.)N[i][0] = - N[i][0];
        if(N[i][1] < 0.)N[i][1] = - N[i][1];
        if(N[i][2] < 0.)N[i][2] = - N[i][2];
    }
else{
    N[i][0] = - N[i][0] ;
    N[i][1] = - N[i][1] ;
    N[i][2] = - N[i][2] ;
}

for( i = 0; i < npolys ; i++) {
    N1[i][0] = 0.0;
    N1[i][1] = 0.0;
    N1[i][2] = 0.0;
    for( j = 0; j < npolys ; j++) {
        N1[P[i]->s[j]][0]+- N1[j][0];
        N1[P[i]->s[j]][1]+- N1[j][1];
        N1[P[i]->s[j]][2]+- N1[j][2];
    }
    for( i = 0; i < npolys ; i++)
        VecNormalize(N1[i]);

VecNegate(a)
Vec a ;
{
    a[0] = 0 - a[0] ;
    a[1] = 0 - a[1] ;
    a[2] = 0 - a[2] ;
}

VecSub(a, b, c)
Vec a, b, c ;
{
    c[0] = a[0] - b[0] ;
    c[1] = a[1] - b[1] ;
    c[2] = a[2] - b[2] ;
}

VecCross(a, b, c)
Vec a, b, c ;
{
    c[0] = a[1] * b[2] - b[1] * a[2] ;
    c[1] = -(a[0] * b[2] - b[0] * a[2]) ;
    c[2] = a[0] * b[1] - b[0] * a[1] ;
}

VecNormalize(a)
Vec a ;
{
    float l ;
    l = (float) sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]) ;
    a[0] /= l ;
    a[1] /= l ;
    a[2] /= l ;
}

float shinyaterial[] = { SPECULAR, 1.0, 0.2, 0.8,
                        EMISSION , 0.8,0.8,0.0,
                        DIFFUSE, 0.8, 0.0, 0.0,
                        AMBIENT, 0.2, 0.0, 0.0,
                        SHININESS, 3.0,
                        LMNULL } ;

float sunl[] = { LCOLOR,1.0,1.0,1.0,
                 POSITION, 0.0, 0.0, 1.0,0.0,
                 LMNULL } ;

```

geomview.c

```

float sun2[] = { LCOLOR,1.0, 0.0,0.0,0,
                POSITION, 0.0, 0.0, 1.0,0.0,
                LNULL };

float sun3[] = { LCOLOR,0.0, 0.0,1.0,
                POSITION, 0.0, 0.0, 1.0,0.0,
                LNULL };

int tum ,tumb1;

DrawObject()
{
    Mouse * mymouse;
    short xmouse,ymouse;
    char str[100];
    int i, j,col_ind ;
    get_surface();
    sprintf(str,"%s %s %s",PATH,"mycedit",ncol,"e");
    if(editPalette->val == 1.0) system(str);
    if(savePosition->val == 1.0) save_position();
    if(getPosition->val == 1.0) get_position();
    if(savePict->val) save_picture();
    if(reset->val) make_reset();
    if(reslight->val) reset_light();
    reshapeviewport();
    c3f(backvec);
    clear();
    zclear();
    MakeVob();
    callobj(Vobj1);
    MakeLight1();
    if(box->val){
        pushmatrix();
        scale(2.0,2.0,2.0);
        Drawbox();
        popmatrix();
    }

    ChangeStack();
    if (model->val == 1) {
        mbind(MATERIAL, 0);
        mbind(LIGHT0, 0);
        mbind(LIGHT1, 0);
        mbind(LIGHT2, 0);
        mbind(LMODEL, 0);
        for (i = 0; i < npolys; i++) {
            c3f(redvec);
            bgnclosedline();
            for (j = 0; j < p[1] -> p_nsides; j++) {
                v3f(p[1] -> p[j]);
            }
            endclosedline();

            if (map->val == 1) {
                mbind(MATERIAL, 0);
                mbind(LIGHT0, 0);
                mbind(LIGHT1, 0);
                mbind(LIGHT2, 0);
                mbind(LMODEL, 0);
                for (i = 0; i < npolys; i++) {
                    bgnpolygon();
                    for (j = 0; j < p[1] -> p_nsides; j++) {
                        col_ind = (int) ((V2[1][j] - mlt[2]) * scal);
                        col_ind = min(ncol-1, max(0, col_ind));
                        c3f((color_palette[ col_ind][0]));
                        v3f(p[1] -> p[j]);
                    }
                    endpolygon();
                }
                if (surface->val || smooth->val) {
                    mbind(MATERIAL, 1);
                    if (light1->val) mbind(LIGHT0, 1); else mbind(LIGHT0, 0);
                    if (light2->val) mbind(LIGHT1, 2); else mbind(LIGHT1, 0);
                    if (light3->val) mbind(LIGHT2, 3); else mbind(LIGHT2, 0);
                    mbind(LMODEL, 1);
                    for (i = 0; i < npolys; i++) {
                        bgnpolygon();
                        for (j = 0; j < p[1] -> p_nsides; j++) {
                            c3f(yelvec);
                            if (surface->val) n3f(N[1]);
                            if (smooth->val) n3f(N1[p[1] -> s[j]]);
                            v3f(p[1] -> p[j]);
                        }
                        endpolygon();
                    }
                }
            }
            swapbuffers();
            make_initpanel();
        }
        def_simple_light_calc()
        {
            lmdef(DEFMATERIAL, 1, 19, shinymaterial);
            lmdef(DEFLIGHT, 1, 10, sun1);
            lmdef(DEFMODEL, 1, 0, NULL);
        }
        use_simple_light_calc()
        {
            mbind(MATERIAL, 1);
            mbind(LIGHT0, 1);
            mbind(LMODEL, 1);
        }
        int exitmode;
        DO_JOB()
        {
            c3f(backvec);
            clear();
            zclear();
            pnl_dopanel();
            reshapeviewport();
            DrawObject();
            exitmode = quit->val;
        }
    }
}

```

```
fclose(fp);
```


6

```

scal=ncol/(maxs[2]-mins[2]);
}
else incol = maxcol; Init_palette();
pnl_drawpanel();
}
get_surface()
{
FILE *fp, *fopen();
fp = fopen("lighting", "r");
if(fp != NULL) {
fscanf(fp, "%f %f %f", &shinymaterial[1], &shinymaterial[2], &shinymaterial[3]);
fscanf(fp, "%f %f %f", &shinymaterial[5], &shinymaterial[6], &shinymaterial[7]);
fscanf(fp, "%f %f %f", &shinymaterial[9], &shinymaterial[10], &shinymaterial[11]);
fscanf(fp, "%f %f %f", &shinymaterial[13], &shinymaterial[14], &shinymaterial[15]);
fscanf(fp, "%f %f %f", &sun1[1], &sun1[2], &sun1[3]);
lndef(DEFMATERIAL, 1, 19, shinymaterial);
lmbind(MATERIAL, 1);
lndef(DEFLIGHT, 1, 10, sun1);
lmbind(LIGHT0, 1);
fclose(fp);
pnl_drawpanel();
}
save_position()
{
FILE *fp, *fopen();
int i;
fp = fopen("pnl_position", "w");
for(i = 0; i < MATRIXSIZE; i++)
fprintf(fp, "%f", OldMatrix[i]);
for(i = 0; i < LIGHTINDEX; i++)
fprintf(fp, "%f", sun1[i]);
for(i = 0; i < LIGHTINDEX; i++)
fprintf(fp, "%f", sun2[i]);
for(i = 0; i < LIGHTINDEX; i++)
fprintf(fp, "%f", sun3[i]);
fprintf(fp, "%f %f %f", light1->val, light2->val, light3->val);
fclose(fp);
}
get_position()
{
int i;
FILE *fp, *fopen();
fp = fopen("pnl_position", "r");
for(i = 0; i < MATRIXSIZE; i++)
for(i = 0; i < LIGHTINDEX; i++)
fscanf(fp, "%f", &OldMatrix[i]);
for(i = 0; i < LIGHTINDEX; i++)
fscanf(fp, "%f", &sun1[i]);
for(i = 0; i < LIGHTINDEX; i++)
fscanf(fp, "%f", &sun2[i]);
for(i = 0; i < LIGHTINDEX; i++)
fscanf(fp, "%f", &sun3[i]);
}
fscanf(fp, "%f", &sun3[1]);
fscanf(fp, "%f %f %f", &light1->val, &light2->val, &light3->val);
fclose(fp);
lndef(DEFLIGHT, 1, 10, sun1);
lmbind(LIGHT0, 1);
lndef(DEFLIGHT, 1, 10, sun2);
lmbind(LIGHT1, 1);
lndef(DEFLIGHT, 1, 10, sun3);
lmbind(LIGHT2, 1);
lmbind(LIGHT3, 1);
loadmatrix(OldMatrix);
pnl_drawpanel();
}
}
MakeVobj()
{
float siz, shx, shy, x_sc, y_sc, z_sc;
float x, y, z;
int thetal;
siz = size -> val;
shx = shix -> val;
shy = shiy -> val;
x_sc = xsc -> val;
y_sc = ysc -> val;
z_sc = zsc -> val;
makeobj(Vobj1 = genobj());
loadmatrix(idmat);
ortho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
closeobj();
makeobj(Vobj = genobj());
callobj(Vobj);
rotate(short)twist -> val, 'y';
rotate(short)theta -> val, 'z';
rotate(short)phi -> val, 'x';
scale(x_sc, y_sc, z_sc);
translate(shx, shy, distance->val);
closeobj();
makeobj(sphere = genobj());
for(thetal = 0; thetal < 360; thetal = thetal+300) {
rotate(300, 'y');
circ(0.0, 0.0, 1.0);
closeobj();
}
MakeLight(xmouse, ymouse);
Screencoord xmouse, ymouse;
long xor, yor;
float wxl, wyl, wzl;
getorigin(&xor, &yor);
xmouse = xmouse - xor;
ymouse = ymouse - yor;
}

```

geomview.c

7

```

main(argc, argv, xmouse, ymouse, &wx1, &wy1, &wz1, &wx2, &wy2, &wz2);
sun1[5] = wx2; sun1[6] = wy2; sun1[7] = 5.0*Max;
lmdf(DEFLIGHT, 1, 10, sun1);
lmbind(LIGHT0, 1);
pushmatrix();
translate(wx2, wy2, wz2);
scale(0.01, 0.01, 0.01);
c3f(4*sun1[1]);
callobj(sphere);
popmatrix();
}

makeLight1()
{
    sun1[5] = xlight1->val;
    sun1[6] = ylight1->val;
    sun1[7] = zlight1->val;
    lmdf(DEFLIGHT, 1, 10, sun1);
    lmbind(LIGHT0, 1);
    sun2[5] = xlight2->val;
    sun2[6] = ylight2->val;
    sun2[7] = zlight2->val;
    lmdf(DEFLIGHT, 2, 10, sun2);
    lmbind(LIGHT1, 2);
    sun3[5] = xlight3->val;
    sun3[6] = ylight3->val;
    sun3[7] = zlight3->val;
    lmdf(DEFLIGHT, 3, 10, sun3);
    lmbind(LIGHT2, 3);
}

save_picture()
{
    char savePict[50], *out_rgb, outStr[20];
    long xor, yor, xsize, ysize;
    getorigIn(&xor, &yor);
    getsize(&xsize, &ysize);
    out_rgb = PNL_ACCESS(TypeIn, inputStr, str);
    sprintf(savePict, "%s %d %d %d", "scrsave ", out_rgb, xor,
        xor+xsize, yor, yor+ysize);
    system(savePict);
}

make_reset()
{
    callobj(Vobj);
    scale(pow(10, size->val), pow(10, size->val), pow(10, size->val));
    getmatrix(OldMatrix);
    .
}

#define RESET(v) v->val = v->initval

reset_light()
{
    RESET(xlight1);
    RESET(ylight1);
    RESET(zlight1);
    RESET(xlight2);
    RESET(ylight2);
    RESET(zlight2);
    RESET(xlight3);
    RESET(ylight3);
    RESET(zlight3);
    pnl_drawpanel();
}

make_initpanel()
{
    RESET(distance);
    RESET(theta);
    RESET(phi);
    RESET(twist);
    RESET(size);
    RESET(shix);
    RESET(shiy);
    RESET(xsc);
    RESET(ysc);
    RESET(zsc);
    pnl_drawpanel();
}

ChangeStack()
{
    callobj(Vobj);
    if(twist->active) rotate((short)twist -> val, 'y');
    if(theta->active) rotate((short)theta -> val, 'z');
    if(phi->active) rotate((short)phi -> val, 'x');
    if(xsc->active) yz->active; if(zsc->active) scale(pow(10, xsc->val),
        pow(10, ysc->val), pow(10, zsc->val));
    if(shix->active) shiy->active; if(distance->active)
        translate(shix->val, shiy->val, distance->val);
    if(size->active) scale(pow(10, size->val), pow(10, size->val), pow(10, size->val));
    multmatrix(OldMatrix);
    getmatrix(OldMatrix);
}

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fp, *fopen();
    int i, j;
    short val;
    int dev;
    ncol = maxcol;
    negflag = 0;
    if (argc == 2)
        negflag = 1;
    init_palette();
    ReadObject();
    keepaspect(1, 1);
}

```

geomview.c

8

```
foreground() ;
wlopen("surface") ;
RGBmode() ;
coublebuffer() ;
gconfig() ;

lsetdepth(0, 0x7FFFFFFF) ;
zbuffer(TRUE) ;

mmode (MVIEWING) ;

Defpanel() ;
pnl_needredraw();
def_simple_light_calc();
callobj(vobj1);
scale(pow(10,size->val),pow(10,size->val));
getmatrix(OldMatrix);

DrawObject();
add_event(wlndget(), REDRAW, ANY, DO_JOB(), 0);
qdevice (REDRAW);
}
```

mycedit.c

```

#include<stdio.h>
#include<device.h>
#include<fcntl.h>
#include<gl.h>
#include<string.h>
#include<fcntl.h>
#define maxcol 10000
#define paramnum 5
#define PALETTE 2
#define LIGHTING 21
#define QUIT 5
#define BITS 31
#define PIECES 32
#define THRASH 33

float blackvec[] = {0.0,0.0,0.0};
float whitevec[] = {1.0,1.0,1.0};
float greenvec[] = {0.0,1.0,0.0};
float bluevec[] = {0.0,0.0,1.0};
float redvec[] = {1.0,0.0,0.0};
float grayvec[] = {0.7,0.7,0.7};
float rgbval[3];

short x,kOld /*number of the changing element of the palette*/
short m; /*number of the changing element of light. param*/
long xo,yo,xsize,ysize;
float color_palette[maxcol][3];
float lightvec[paramnum][3],shine;
long lgid,exitmode, mode,moda;
int menu,menu1,menu2,menu3,submenu1,ncolmenu,ncolmenu1;
float range,rgbvalWidth,xRectSlide,delta,dx,dx2,side;
int ncol,index;

int_range(n)
int n;
{
    mode = PALETTE;
    ncol = n;
    init_palette(ncol);

    change_range(n)
    int n;

    .at ncol_old;
    float new_palette[maxcol][3];
    .nt i,j,l;
    int coef;

    mode = PALETTE;
    ncol_old = ncol;
    ncol = n;
    .f(ncol_old < ncol){
        coef = ncol/ncol_old;
        for(j = 0;j<ncol_old;j++){
            for(i = 0;i<coef;i++){
                for(l = 0;l<3;l++){
                    if(mods == PIECES)
                        new_palette[j*coef+i][l] = color_palette[j][l];
                    if(mods == PIECES)
                        new_palette[j*coef+i][l] = color_palette[j][l];
                }
            }
        }
    }
    .f(ncol_old > ncol){
        coef = ncol_old/ncol;
        for(j = 0;j<ncol;j++){
            for(i = 0;i<coef;i++){
                for(l = 0;l<3;l++){
                    if(mods == BITS)
                        new_palette[j*coef+i][l] = color_palette[j][l] + (1 - (color_palette[j*coef+i][l])/coef) *
                            color_palette[j][l];
                    if(mods == BITS)
                        new_palette[j*coef+i][l] = color_palette[j][l];
                }
            }
        }
    }
    for(j = 0;j<ncol;j++){
        for(i = 0;i<3;i++){
            color_palette[j][i] = new_palette[j][i];
        }
    }
    if(ncol_old>ncol){
        coef = ncol_old/ncol;
        for(j = 0;j<ncol;j++){
            for(i = 0;i<3;i++){
                new_palette[j][i] = color_palette[j*coef+i][l];
            }
        }
        for(j = 0;j<ncol;j++){
            for(i = 0;i<3;i++){
                color_palette[j][i] = new_palette[j][i];
            }
        }
    }
}

main(argc,argv)
int argc;
char *argv[];
{
    FILE *fp1,*fp2,*fopen();

    ncol = atoi(argv[1]);
    initialize();
    init_palette(ncol);
    DrawObj();
    while(!exitmode){
        DrawObj();
        swapbuffers();
    }
    processinput();
    winclose(lgid);
}

initialize()
{
    short i;
    keepaspect(1,1);
    foreground();
    lgid = winopen("color_edit");
    RGBmode();
    doublebuffer();
    gconfig();
    ortho(0.0,1.0,0.0,1.0,-1.0,1.0);
    c3f(whitevec);
    clear();
    qdevice(RIGHTMOUSE);
    qdevice(LEFTMOUSE);
}

```

mycedit.c

```

for(i=0;i<3;i++)
    rgbval[i] = 0.0;
x = -1;
mode = PALETTE;
moda = THRASH;
index = 0;
MyMenu();
t1e(RIGHTHOUSE, MOUSEX, MOUSEY);
t1e(LEFTHOUSE, MOUSEX, MOUSEY);
getorigln(xo, y0);
getsize(xsize, ysize);
}

processinput()
{
    short val;
    short mx, my;
    long menuval;
    short i;

    while(!qtest()) {
        DrawObj();
        switch(qread(&val)) {
            case (REDRAW):
                reshapeviewport();
                getorigln(xo, y0);
                getsize(xsize, ysize);
                while(!qtest()) {
                    DrawObj();
                    swapbuffers();
                }
                break;
            case (LEFTHOUSE):
                if(val) {
                    qread(&mx);
                    qread(&my);
                    make_rgbval(mx, my);
                    DrawObj();
                    swapbuffers();
                }
                break;
            case (RIGHTHOUSE):
                menuval = dupup(menu1);
                switch(menuval) {
                    case 1:
                        if (mode == PALETTE) init_palette(ncol);
                        else init_lighting();
                        DrawObj();
                        swapbuffers();
                        break;
                    case 2:
                        if (mode == PALETTE) get_palette();
                        else get_lighting();
                        DrawObj();
                        swapbuffers();
                        break;
                    case 3:
                        if (mode == PALETTE) save_palette();
                        else save_lighting();
                        DrawObj();
                }
            }
        }
    }

    swapbuffers();
    break;
case LIGHTING:
    mode = LIGHTING;
    init_lighting();
    DrawObj();
    swapbuffers();
    break;
case BITS:
    mode = PALETTE;
    moda = BITS;
    DrawObj();
    swapbuffers();
    break;
case PIECES:
    mode = PALETTE;
    moda = PIECES;
    DrawObj();
    swapbuffers();
    break;
case THRASH:
    mode = PALETTE;
    moda = THRASH;
    DrawObj();
    swapbuffers();
    break;
case 41:
    if (mode == PALETTE)
        save_palette();
    else save_lighting();
    exitmode = 1;
    break;
case 42:
    exitmode = 1;
    break;
case 43:
    exitmode = 0;
    break;
default:
    break;
}

init_palette(ncol)
int ncol;
{
    int i, j;
    float dx1;
    for(i = 0; i < 3; i++)
        for(j = 0; j < ncol; j++)
            color_palette[j][i] = 0.0;
}

```

mycedit.c

```

void = -1;
index = 0;
for (i = 0; i < 3; i++) {
    dx1 = 4./ncol;
    if (i == 0) {
        for (j = ncol; j >= (int) (3*ncol/4)+1; j--) {
            color_palette[j][i] = 1.0;
        }
        for (j = (int) (3*ncol/4); j >= ncol/2; j--) {
            color_palette[j][i] = dx1*j - (int) (ncol/2);
        }
    }
    if (i == 1) {
        for (j = (int) (ncol); j >= (int) (3*ncol/4); j--) {
            color_palette[j][i] = 1 - dx1*j - (int) (3*ncol/4);
        }
        for (j = (int) (3*ncol/4); j >= (int) (ncol/4); j--) {
            color_palette[j][i] = 1.0;
        }
    }
    for (j = (int) (ncol/4); j >= 0; j--) {
        color_palette[j][i] = dx1*j;
    }
}

if (i == 2) {
    for (j = ncol/2; j >= (int) (ncol/4); j--) {
        color_palette[j][i] = 1.0 - dx1*j - ncol/4;
    }
    for (j = (int) (ncol/4); j >= 0; j--) {
        color_palette[j][i] = 1.0;
    }
}

/* draw RGB rects */
xRectSide = 1./5. - delta;
yRectSide = 1. - 4*delta;
for (i = 1; i < 4; i++) {
    c3f(grayvec);
    rectf(delta*i+xRectSide*(i-1), 3*delta, delta*i+xRectSide*i, 3*delta+yRectSide);
    c3f(whitevec);
    rectf(delta*(i+1)+xRectSide*(i-1), 4*delta, delta*(i-1)+
        xRectSide*i, 2*delta+yRectSide);
    c3f(blackvec);
}

range = yRectSide - 2*delta;
rgbvalSide = range/100.;
rgbvalWidth = xRectSide - 2*delta;
rgbvalHeight = range*rgbval[i-1]+4*delta;
for (k1 = 0; k1 < 3; k1++)
    col[k1] = 0.0;
col[i-1] = 1.0;
c3f(col);
rectf(delta*(i+1)+xRectSide*(i-1), rgbvalHeight, delta*(i+1)+xRectSide*(i-1) +
    rgbvalWidth, rgbvalHeight-rgbvalSide);
rect(delta*i+xRectSide*(i-1), 3*delta, delta*i+xRectSide*i, 3*delta+yRectSide);
rect(delta*(i+1)+xRectSide*(i-1), 4*delta, delta*(i-1)+xRectSide*i, 2*delta+yRectSide);
}

```

```

*sample rectangle*/
c3f(rgbval);
rectf(1 - delta - 2*xRectSide, 3*delta, 1 - delta, 3*delta+yRectSide);
c3f(blackvec);
rect(1 - delta - 2*xRectSide, 3*delta, 1 - delta, 3*delta+yRectSide);
)

ake_rgbval(x,y)
short x,y;

float z1,z2;
int b1,b2;
int ncol_old, jump;
float xl,y1,y2; short i,OK,j;
OK = 0;

xl = (float)(x-x0)/(float)xsize;
y1 = (float)(y-y0)/(float)ysize;

if(y1 > 2*delta) {
    y1 = (y1 - (4*delta))/range;
    i = (int)(xl/delta + xRectSide);
    if((xl - (delta*(i+2) + xRectSide*1)) > 0 && (xl - (delta*(i+2) + xRectSide*1))
    < rgbvalWidth) OK = 1;
    if(y1 > 0 && y1 < 1.0 && OK == 1) rgbval[i] = y1;
}
/*choosing k - the el of the palette*/
else {if(mode == PALETTE) {
    if(y1 > delta && y1 < 2*delta && x1 > delta && x1 < 1.0 - delta) {
        kOld = k;
        k = (int)((xl - delta)/dx);
        index++;
        printf("%d k %d kOld\n", k, kOld);
        if(mode == PIECES && index%2 == 0) {
            if(kOld < k && k >= 0 && kOld >= 0) for(j = kOld; j < k+1; j++)
                for(i = 0; i < 3; i++)
                    color_palette[j][i] = rgbval[i];
            else if(k >= 0 && kOld >= 0) for(j = k; j < kOld+1; j++)
                for(i = 0; i < 3; i++)
                    color_palette[j][i] = rgbval[i];
        }
        if(mode == THRASH && index%2 == 0) {
            ncol_old = ncol;
            if(k > kOld) {
                z2 = (float)ncol_old/(float)k;
                z1 = (float)ncol_old/(float)kOld;
                jump = k-kOld;
            }
            if(k < kOld) {
                z1 = (float)ncol_old/(float)k;

```

mycedit.c

5

```

{
    int j;
    FILE *fp, *fopen();

    fp = fopen("color_palette", "w");
    fprintf(fp, "%d\n", ncol);
    for(j = 0; j < ncol; j++)
        fprintf(fp, "%f %f %f\n", color_palette[j][0], color_palette[j][1],
            color_palette[j][2]);
    fclose(fp);
}

get_lighting()
{
    int j;
    FILE *fp, *fopen();
    fp = fopen("lighting", "r");
    if(fp != NULL) {
        for(j = 0; j < paramnum; j++)
            fscanf(fp, "%f %f %f", &lightvec[j][0], &lightvec[j][1],
                &lightvec[j][2]);
        k = -1;
        fclose(fp);
    }
    else init_lighting();
    k = -1;
}

save_lighting()
{
    int j;
    FILE *fp, *fopen();
    fp = fopen("lighting", "w");
    for(j = 0; j < paramnum; j++)
        fprintf(fp, "%f %f %f\n", lightvec[j][0], lightvec[j][1],
            lightvec[j][2]);
    fclose(fp);
}

init_lighting()
{
    /*SPECIAL*/
    lightvec[0][0] = 1.0; lightvec[2][0] = 1.0;
    lightvec[0][1] = 0.0; lightvec[2][1] = 0.6;
    lightvec[0][2] = 0.0; lightvec[2][2] = 0.0;
    /*EMISSION*/
    lightvec[1][0] = 1.0; lightvec[3][0] = 1.0;
    lightvec[1][2] = 0.2; lightvec[3][1] = 0.0;
    lightvec[1][3] = 0.0; lightvec[3][2] = 0.0;
    /*LIGHT*/
    lightvec[4][0] = 1.0;
    lightvec[4][1] = 1.0;
    lightvec[4][2] = 1.0;
}

MyMenus()
{
    menu2 = defpup("SAVE? %t YES %x41 NO %x42|CANCEL %x43 ");
    ncolmenu = defpup("COLOR RANGE %t %f18 %x8116 %x16132 %x32164 %x641256 %x256", init_range);
    ncolmenu1 = defpup("COLOR RANGE %t %f18 %x8116 %x16132 %x32164 %x641256 %x256", change_range);
    submenu1 = defpup("PALETTE %t|BITS %x311 PIECES %x321 THRASH %x33");
    menu = defpup("EDIT %t|LIGHTING %x21|PALETTE %m", submenu1);
    addtopup(menu, "COLOR RANGE %m", ncolmenu);
    menu1 = defpup("EDIT %t|INIT %m", ncolmenu);
    addtopup(menu1, "GET|SAVE|CHANGE MODE %m", menu);
    addtopup(menu1, "QUIT %m", menu2);
}

```


``` prepar.c ```

```

#include<stdio.h>
#include<device.h>
#include<fcntl.h>
#include<gl.h>
#include<string.h>
#include<cpack.h>
#include<panel.h>

#define N 1 /*number of buttons on the control panel*/
#define
    v = pnl_mkact(pnl_button); \
    v -> label = n; \
    v->labeltype = PNL_LABEL_BOTTOM; \
    v -> minval = b; \
    v -> maxval = t; \
    v -> val = 1; \
    v -> w = 1.0; \
    v -> h = 0.5; \
    v -> x = x; v -> y = y
    **;

Actuator *inputStr;
anel *MyDialBox(labelStr, Button, butAct)
char *labelStr, *Button[N];
ctuator *butAct[N];

nt i, n;
ctuator *mylabel;
loat sep, x, y;
anel *panelPtr;
ep = 0.5;
anelPtr = pnl_mkpanel();
anelPtr->x = 600;
anelPtr->y = 500;
anelPtr->visible = TRUE; /* panel is initialised invisible */
/* to make it visible say panelPtr->visible = TRUE */
/* and pnl(fixpanel(panelPtr)); */

= 0;
or(i = 0; i < N; i++)
{
    Button[i] = 0;
    Button[i] = Button[i];
    **;

    = 2*sep;
    = 0.0;
    putStr = pnl_mkact(pnl_typeIn);
    _l_addact(inputStr, panelPtr);
    = 12*sep;
    = inputStr->y - 2*sep;
    or(i = 0; i < N; i++)
    {
        JTON(butAct[i], Button[i], 0.0, 1.0, 0.0);
        = sep; butAct[i] -> w;
        _l_addact(butAct[i], panelPtr);

        /label = pnl_mkact(pnl_label);
        /label->label = labelStr;
        /label->y = inputStr->y + 2*sep;
        /label->x = sep;
        _l_addact(mylabel, panelPtr);
        xturn panelPtr;

    in()

```

```

{
    FILE *fp, *fopen();
    char geodata[100], plus[100];
    char *mystring;
    int i;
    long lgidl;
    short mx, my;
    char *But[N];
    Actuator *but[N];
    Panel *dial;
    int mode = 1;
    float x = 0.0; float y = 0.0;
    foreground();
    noport();
    keepaspect(1, 1);
    lgidl = winopen("");
    doublebuffer();
    RGBmode();
    gconfig();
    But[0] = "CONTINUE";
    But[1] = "SELECT";
    dial = MyDialBox("INPUT FILE NAME", But, but);
    pnl_needredraw();
    fp = fopen("geoname", "w"); /*erase geoname*/
    fclose(fp);
    while('but[0] -> val){
        pnl_userredraw();
        pnl_dopanel();
        mystring = PNL_ACCESS(TypeIn, inputStr, str);
    }
    pnl_delpanel(dial);
    fp = fopen("geoname", "w");
    fprintf(fp, "%s", mystring);
    fclose(fp);

    /*winclose(lgidl);*/
}

```

sgeomview.c

```

#include <stdio.h>
#include <math.h>
#include <gl.h>
#include <panel.h>
#include <device.h>
#define ncol 12
#define k 3.5
#define ANY -1
#define SIZE 100000

float legend[ncol][3] = {
    {0.0, 0.0, 0.8},
    {0.0, 0.0, 1.0},
    {0.0, 0.6, 0.6},
    {0.0, 0.8, 0.0},
    {0.0, 1.0, 0.0},
    {0.2, 0.9, 0.0},
    {1.0, 1.0, 0.0},
    {1.0, 0.8, 0.0},
    {1.0, 0.6, 0.0},
    {1.0, 0.4, 0.0},
    {1.0, 0.2, 0.0},
    {1.0, 0.0, 0.0},
};

typedef float Vec[3];

typedef struct {
    int p_nsides;
    Vec *p;
    int *s;
} Polygon;

int negflag;
/*float backvec[] = {0.8, 0.8, 0.8};*/
float backvec[] = {0.0, 0.0, 0.0};
float blackvec[] = {0.0, 0.0, 0.0};
float redvec[] = {1.0, 0.0, 0.0};
float whitevec[] = {1.0, 1.0, 1.0};

float idmat[] = {1.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0};

Polygon *
MakePolygon(nsides)
int nsides;
{
    Polygon * p;
    p = (Polygon *) malloc (sizeof(Polygon));
    p->p_nsides = nsides;
    p->p = (Vec **) calloc(nsides, sizeof(Vec *));
    p->s = (int *) malloc(3* sizeof(int));
    return p;
}

Vec
V[SIZE];
Polygon * P[SIZE];

Vec
n[SIZE], n1[SIZE];
int
npoints, npolys;
Vec
mins = {1000000, 1000000, 1000000};
Vec
maxs = {-1000000, -1000000, -1000000};
float
m1 = 1000000;
float
ma = -1000000;
float
V1[SIZE], V2[SIZE][3], V3[SIZE][3];

#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

float Max, scal, fovy;

#define MKHSLIDER(v, n, b, t, i) \
    v = pnl_mkact(pnl_hslider); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    pnl_addact(v, p); \
    y += v->h + sep

#define MKHSLIDER1(v, n, b, t, i) \
    v = pnl_mkact(pnl_hslider); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hl; v->w = w; \
    pnl_addact(v, p); \
    x += v->w + sep1

#define BUTTON(v, n, b, t, i) \
    v = pnl_mkact(pnl_radio_button); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    pnl_addact(v, p); \
    y += v->h + sep; \
    x += v->w - 4*sep-w

#define BUTTON1(v, n, b, t, i) \
    v = pnl_mkact(pnl_radio_button); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hl; v->w = w; \
    pnl_addact(v, p); \
    x += v->h + 3*sep

#define BUTTON2(v, n, b, t, i) \
    v = pnl_mkact(pnl_toggle_button); \
    v->label = n; \
    v->minval = b; \
    v->maxval = t; \
    v->val = i; \
    v->x = x; v->y = y; \
    v->h = hl; v->w = w; \
    pnl_addact(v, p);

```

sgemview.c

2

```

y += v -> h + sep; \
x += v -> w - 4*sep-w

Actuator * distance, * theta, * phi, * model, * surface, * map,
* twist, * angle, * shlx, * shly, * smooth, * xsc, * ysc, * tumble;

DefPanel ()
{
    Panel * p;
    float x = 0.0, y = 0.0, sep = 0.7, sepl=1.1;
    float hl= 0.5, w = 1.0;

    p = pnl_mkpanel();
    p -> label = "Control Panel v.10";
    p -> ppu = 36.0;

    MKHSLIDER(distance, "DISTANCE", k*Max, k*k*Max, k*Max);
    MKHSLIDER(theta, "2 ROTATION", 0.0, 3600.0, 0.0);
    MKHSLIDER(phi, "X ROTATION", 0.0, 3600.0, 0.0);
    MKHSLIDER(twist, "Y ROTATION", 0.0, 3600.0, 0.0);
    MKHSLIDER(angle, "SIZE", 0.5, 0.01, 0.3);
    MKHSLIDER(shlx, "SHIFT_X", -1.0*Max, Max, 0.0);
    MKHSLIDER(shly, "SHIFT_Y", -1.0*Max, Max, 0.0);
    MKHSLIDER(xsc, "XSCALE", 1.10, 1.0);
    MKHSLIDER(ysc, "YSCALE", 1.10, 1.0);
    BUTTON1(model, "Model", 0.0, 1.0, 0.0);
    BUTTON1(surface, "Surface", 0.0, 1.0, 0.0);
    BUTTON1(map, "Map", 0.0, 1.0, 0.0);
    BUTTON1(smooth, "Smooth", 0.0, 1.0, 0.0);
    BUTTON2(tumble, "Tumble", 0.0, 1.0, 0.0);

    float shiftx, shifty, tempo;

    ReadObject ()
    {
        int i, j, ku, nverts, index[3], nsides;
        Vec tmp1, tmp2;

        scanf("%d %d\n", & npolys, & npoints);

        fprintf(stderr, "%d points, %d polys\n", npoints, npolys);

        for (j = 0; j < 3; j++) {
            mins[j] = lelo;
            maxs[j] = -lelo;
        }

        for (i = 0; i < npoints; i++) {
            scanf("%f %f %f\n", &V[i][0], &V[i][1], &V[i][2]);
        }

        /*Lala: I changed here */
        for (i = 0; i < npoints; i++) {
            tempo=V[i][1];
            V[i][0] = V[i][0] - shiftx;
            V[i][1] = tempo - shifty;
        }

        for (i = 0; i < npolys; i++) {
            P[i] = MakePolygon(3);
            for (j = 0; j < 3; j++) {
                scanf("%d", &index[j]);
                P[i]->s[j] = index[j] - 1;
                P[i]->p[j] = v + (index[j] - 1);
                V2[i][j] = V[index[j] - 1][2];
            }
            for (ku = 0; ku < 3; ku++) {
                mins[ku] = min(mins[ku], V[index[j]-1][ku]);
                maxs[ku] = max(maxs[ku], V[index[j]-1][ku]);
            }
            V3[i][0] += (V[index[j]-1][ku] + V[index[j]-1][ku]);
            V3[i][1] = sqrt(V3[i][0]);
            m1 = min(m1, V3[i][1]);
            ma = max(ma, V3[i][1]);
        }
        V1[i] = V2[i][0] + V2[i][1] + V2[i][2];
        V1[i] = V1[i]/3.;

        /*
        fprintf(stderr, "bbox: %f - %f\n", mins[0], maxs[0]);
        fprintf(stderr, "bby: %f - %f\n", mins[1], maxs[1]);
        fprintf(stderr, "bbz: %f - %f\n", mins[2], maxs[2]);
        */
        if (ma >= m1)
            scal=ncol/(ma-m1);
        else scal = ncol/ma;

        Max = max(maxs[0]-mins[0], maxs[1]-mins[1]);
        Max = max(Max, maxs[2]-mins[2]);
        shiftx = 0.5*(maxs[0]-mins[0]);
        shifty = 0.5*(maxs[1]-mins[1]);

        for (i = 0; i < npolys; i++) {
            VecSub(P[i]->p[0], P[i]->p[1], tmp1);
            VecSub(P[i]->p[1], P[i]->p[2], tmp2);
            VecCross(tmp1, tmp2, N[i]);
            VecNormalize(N[i]);
            if (negflag) { else
                N[i][0] = -N[i][0];
                N[i][1] = -N[i][1];
                N[i][2] = -N[i][2];
            }

            for (l = 0; l < npolys; l++) {
                for (j = 0; j < 3; j++) {
                    N1[i][0] = 0.0;
                    N1[i][1] = 0.0;
                    N1[i][2] = 0.0;
                }

                for (l = 0; l < npolys; l++) {
                    for (j = 0; j < 3; j++) {
                        N1[P[i]->s[j]][0] += N[i][0];
                        N1[P[i]->s[j]][1] += N[i][1];
                        N1[P[i]->s[j]][2] += N[i][2];
                    }
                }
                for (l = 0; l < npolys; l++)
                    VecNormalize(N1[i]);
            }
        }
    }
}

```

geomview.c

3

```

perspective(fovy, 1.0, 0.1, k*k* Max*(1./ang)) ;
polarview(distance->val,0.0,0.0,0.0) ;
rotate((short)twist -> val,'y') ;
rotate((short)theta -> val,'y') ;
rotate((short)phi -> val,'x') ;
scale(x_sc,y_sc,1.0) ;
rotate(tum,'y') ;

/* rotate (900,'z') ;*/

translate(shx,shy,0.0) ;
DrawBox() ;
translate(-1.0*shiftx,-1.0*shifty,0.0) ;
if (templ == 1) {
    lmbind(MATERIAL, 0) ;
    lmbind(LIGHT0, 0) ;
    lmbind(LMODEL, 0) ;
    for (i = 0 ; i < npolys ; i++) {
        c3f(redvec) ;
        /*c3f(blackvec) ;*/
        bgnclosedline() ;
        for (j = 0 ; j < p[i] -> p_nsides ; j++) {
            v3f(p[i]->p[j]) ;
        }
        endclosedline() ;
    }
}

if (temp3 == 1) {
    lmbind(MATERIAL, 0) ;
    lmbind(LIGHT0, 0) ;
    lmbind(LMODEL, 0) ;

    for (i = 0 ; i < npolys ; i++) {
        col_ind = (int) ((v1[i]-mins(2))*scal) ;
        col_ind = min(ncol-1,max(0,col_ind)) ;
        c3f(legend[ col_ind[0] ]) ;
    }
    bgnpolygon() ;
    for (j = 0 ; j < p[i] -> p_nsides ; j++) {
        col_ind = (int) ((v3[i][j]-mi)*scal) ;
        col_ind = min(ncol-1,max(0,col_ind)) ;
        c3f(legend[ col_ind[0] ]) ;
        v3f(p[i]->p[j]) ;
    }
    endpolygon() ;
}

if (temp2 || temp0) {
    lmbind(MATERIAL, 1) ;
    lmbind(LIGHT0, 1) ;
    lmbind(LMODEL, 1) ;
    for (i = 0 ; i < npolys ; i++) {
        bgnpolygon() ;
        for (j = 0 ; j < p[i] -> p_nsides ; j++) {
            c3f(legend[ 71(0) ]) ;
            if (temp2) n3f(N(1)) ;
            if (temp0) n3f(N1(p[i]->s[j])) ;
            v3f(p[i]->p[j]) ;
        }
        endpolygon() ;
    }
}

float l ;
l = (float) sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]) ;

a[0] /= l ;
a[1] /= l ;
a[2] /= l ;

int tum, tumbl ;
DrawObject() ;

int i, j, col_ind ;
float ang, shx, shy, x_sc, y_sc ;
templ=0 ;
temp2=surface -> val ;
temp0=smooth -> val ;
temp3=map -> val ;
ang = angle -> val ;
shx = shx -> val ;
shy = shy -> val ;
x_sc = x_sc -> val ;
y_sc = y_sc -> val ;

reshapeviewport() ;
c3f(backvec) ;
clear() ;
zclear() ;
fovy=atan(ang)*1800/PI ;
fovy=2*fovy ;

loadmatrix(ldmat) ;

```

sgomview.c

4

```

    }
    swapbuffers();
}

float shinyaterial[] = { SPECULAR, 1.0, 0.0, 0.0,
    EMISSION, 1.0, 20.0, 0.0,
    DIFFUSE, 1.0, 0.6, 0.0,
    AMBIENT, 1.0, 0.0, 0.0,
    SHININESS, 3.0,
    LMNULL };

float sun[] = { POSITION, 0.0, 0.0, 1.0, 0.0,
    LMNULL };

def_simple_light_calc()
{
    lmdef(DEFMATERIAL, 1, 19, shinyaterial);
    lmdef(DEFLIGHT, 1, 6, sun);
    lmdef(DEFLMODEL, 1, 0, NULL);
}

use_simple_light_calc()
{
    lmbind(MATERIAL, 1);
    lmbind(LIGHT0, 1);
    lmbind(LMODEL, 1);
}

Osl()
{
    float ce;
    c3f(reduce);
    ce = (maxs[2]-mins[2])/2;
    move(0.0, 0.0, ce);
    draw(1.0, 1.0, ce);

    /*cmov (-0.6, 0.499, ce);
    charstr("X");*/
    move(-0.5, 0.5, ce);
    draw(-0.5, 0.6, ce);
    /*cmov (-0.499, 0.6, ce);
    charstr("Y");*/
    move(-0.5, 0.5, ce);
    draw(-0.5, 0.5, ce-0.1);
    /*cmov (-0.5, 0.499, ce-0.1);
    charstr("Z");*/

    DO_JOB()
    .nt mm=0;
    num=80;
    c3f(backvec);
    clear();
    zclear();
    for(;;)
    {
        while (pnl_dopanel());
        reshapeviewport();
        DrawObject();
        Osl();
        tumble = tumble -> val;
        if(tumb: == 1)
            tum += .1m;
    }
}

main(argc, argv)
int argc;
char * argv[];
{
    int i, temp2;
    short val;
    int dev;

    if (strchr(argv[0], 'n'))
        negflag = 0;
    ReadObject();

    keepaspect(1, 1);
    foreground();
    winopen("surface");
    RGBmode();
    doublebuffer();
    geonfig();

    lsetdepth(0, 0x7FFFFFFF);
    zbuffer(TRUE);

    mmode(MVIEWING);

    DefPanel();
    pnl_needredraw();
    def_simple_light_calc();
    DrawObject();
    add_event(winget(), REDRAW, ANY, DO_JOB(), 0);
    qdevice(REDRAW);
}

float kvadrat1[] = {-1.0, -1.0, -1.0};
float kvadrat2[] = {-1.0, 1.0, -1.0};
float kvadrat3[] = {1.0, 1.0, -1.0};
float kvadrat4[] = {1.0, -1.0, -1.0};

Drawbox()
{
    char string1[5], string2[5], string3[5], string4[5];

    sprintf(string1, "%s", "X");
    sprintf(string2, "%s", "Y");
    sprintf(string3, "%s", "Z");
    sprintf(string4, "%s", "O");
    c3f(whitevec);

    cmov(1.0, -0.97, -1.0);
    charstr(string1);
    cmov(-1.0, 1.03, -1.0);
    charstr(string2);
    cmov(-1.03, -1.03, 1.03);
    charstr(string3);
    cmov(-0.97, -0.97, -0.97);
    charstr(string4);
}

```

sgeomview.c

```

pushmatrix();
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

pushmatrix();
translate(0,0,0,0,2,0);
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

pushmatrix();
rotate(-90,'y');
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

pushmatrix();
translate(-2,0,0,0,0,0);
rotate(-90,'y');
bgnclosedline();
v3f(kvadrat1);
v3f(kvadrat2);
v3f(kvadrat3);
v3f(kvadrat4);
endclosedline();
popmatrix();

```

adds.f

1

```

subroutine adds(fnam, mad.t, ind5, ind6)
  tpiahc'qlqciq opophyx to-ek i doablehie dualxhyx to-ek
  parameter(maxst=70000, maxtr=2*maxst)
  parameter(rvmin=1+1e-5)
  integer next, sites, tgl
  'nst' = 0 to-ek, 'r' = ix koopdihaty
  'ntr' = 0 tpeugolxhik'b, '/net/' - cplcok smevhocti dliq
  qpafa dualxhgo k tpiahguliqci
  common /net/ ntr, next(3, maxtr), sites(3, maxtr)
  common /cones/ vec(3, maxtr)
  common /bld/ radv(maxtr), lord(maxtr)
  common/externum/rmin, rmax, radius, rad_ex
  logical ok
  logical frametr
  character*40 fnam
  format(a40)
  open(1, file=fnam)
  open(2, file='stempor', form='unformatted', status='unknown')
  open(3, file='saddual', form='unformatted', status='unknown')
  open(7, file='triang')
  open(8, file='text')
  nst=0
  1045 continue
  nst=nst+1
  read (1, *, end=1020, err=1020) (r(i0, nst), i0=1, 4)
  goto 10
1020 ns=nst-1
  rmin = 1.e50
  rmax = 1.e50
  do 788 i = 1, nst
    rmin = min(rmin, r(4, i))
    rmax = max(rmax, r(4, i))
  788 continue
  'cc print *, ' ----', nst, ' points have been read <---'
  if(nst.ge.maxst) then
    print *, ' ----> too much for arrays been declared <---'
    stop 12
  end if
  call tetra
  ha-. kohlfiqpaciq - tetpaldp c bep(.haml i, 2, 3, 4)
  tgl=1
  kkk=1000
  do 20 np=5, nst
    if(nst.ge.kkk.and.mod(np, kkk).eq.0) then
      print *, ' ----> np, % points have been included <---'
      end if
      bxl'-itx to-ku 'np' b tpiahguliqci', dliq lto go
      hajti tp-k 'tgl', b kotopom levit 'np'
      call search(tgl, np, ok, mk)
      if(ok) then
        cdelatx bapicentpi-ekoe podpazdelehie
        call bar(tgl, np, mk)
        pepectoltx tpiahguliqci'
        call recon(tgl)
        end if
      20 continue
      print *, ' triangulation is constructed'
      if(ind6.eq.1) then
        call graphics(lgid)
        call output1
        end if
        if(ind6.eq.0) then
          call output
          end if
          if(nst.ge.madst) then
            write(8, *) '40 60'
            write(8, *) 'triangulation of initial points is constructed'
            write(8, *) 'the second input parameter is too small'
            write(8, *) 'interpolation is not aloud'
            close(8)
            call try1('text')
            close(3)
            close(1)
            ind5 = 1
            if(ind6.eq.1) call winclo(lgid)
            return
            end if
            ind5 = 0
            nst0=nst
            ic=0
            1040 do 30 i2=1, ntr
              radv(i2)=scal(vec(1, i2), vec(1, i2))
              continue
              call kb07a(radv, ntr, lord)
              print *, 'max rad vec =', radv(ntr)
              it=ntr
              if(nst.eq.madst.or.it.lt.2*ntr/3.or.
                radv(it).lt.rvmin*2) goto 1060
              rv=sqrt(scal(vec(1, lord(it)), vec(1, lord(it))))
              if(rv.lt.rvmin) then
                it=it-1
                goto 1050
              end if
              do 40 j1=1, 3
                r(j1, nst+1)=vec(j1, lord(it))/rv
                continue
                tgl=lord(it)
                call search(tgl, nst+1, ok, mk)
                if(ok.and..not.frametr(tgl, r, sites)) then
                  r(4, nst+1)=(r(4, sites(1, tgl))+r(4, sites(2, tgl))
                    +r(4, sites(3, tgl)))/3
                  call bar(tgl, nst+1, mk)
                  call recon(tgl)
                  nst=nst+1
                  if(mod(nst-nst0, 1000).eq.0) then
                    print *, ' ----> nst-nst0,
                      ' dual points have been added <---'
                    end if
                  end if
                  it=it-1
                  if(ntr.gt.maxtr) then
                    print *, ' maxtr is too small'
                    stop 13
                    end if
                  goto 1050
                continue
                print *, ' # iterations =', ic
                if(it.lt.ntr.and.nst.lt.maxst) goto 1040
              1060 continue
              ccc
            end if

```

adds.f

```

ccc print ' ', '----> ok! i write ... <----'
ccc print ' ', '----> # pnt =', nst, ' <----'
write (3) ntr, nst
write (3) ((r(12,13), i2=1,4), i3=1, nst)
write (3) ((sites(12,13), i2=1,3), i3=1, ntr)
close(3)
if (ind6.eq.1) call winclo(tigid)
return
end

subroutine tetra
* initializaci q dq cfepl-eko j tpiaghulqcll
parameter(maxst=70000, maxtr=2*maxst)
parameter(e=1e-6
, el=0.1)
integer next, sites, tgl
common /points/ nst, r(4, maxst)
common /net/ ntr, next(3, maxtr), sites(3, maxtr)
logical flag
save

if (nst.lt.4) then
print ' ', '----> data are wrong <----'
stop 21
end if

* bektop 'r' dolveh imetx edihl-hu --'hu
flag=.false.
do 10 n1=1, nst
rad=scal(r(1,n1), r(1,n1))
if (abs(rad-1).gt.e) then
print ' ', ' bektop r dolveh imetx diihu = 1'
print '(16, " /r/ =', q18.9)', n1, rad
flag=.true.
end if
if (r(4,n1).le.0) then
print ' ', ' radiu-bektop dolveh bytx polovitelxhym'
print '(16, " /r(4, " , i4, " ) =', q18.9)', n1, r(4,n1)
flag=.true.
end if
10 continue
if (flag) stop 22

* ha-- konfigupaci q - tetpaldp c bep(iham1 1,2 3, i
ntr=4
next(1,1)=4
next(2,1)=2
next(3,1)=3
next(1,2)=4
next(2,2)=3
next(3,2)=1
next(1,3)=4
next(2,3)=1
next(3,3)=2
next(1,4)=1
next(2,4)=3
next(3,4)=2
sites(1,1)=4
sites(2,1)=2
sites(3,1)=3
sites(1,2)=4
sites(2,2)=3

```

```

sites(3,2)=1
sites(1,3)=4
sites(2,3)=1
sites(3,3)=2
sites(1,4)=1
sites(2,4)=3
sites(3,4)=2
* ha-alo koopdihat dolvho levatx bhutpi tetpaldp
1010 do 20 n2=2, nst
t1=scal(r(1,1), r(1,n2))
if (t1.lt.-.2.and.t1.gt.-.8) goto 1020
20 continue
print ' ', '----> Initial data are wrong <----'
print ' ', '----> stop <----'
stop 23
1020 continue
do 30 j0=1,4
u1=r(j0,2)
r(j0,2)=r(j0,n2)
r(j0,n2)=u1
30 continue
do 40 n3=3, nst
t2=volum(r(1,2), r(1,1), r(1,n3))
if (abs(t2).gt..2) goto 1030
40 continue
print ' ', '----> Initial data are wrong <----'
print ' ', '----> stop <----'
stop 24
1030 continue
do 50 j1=1,4
u2=r(j1,3)
r(j1,3)=r(j1,n3)
r(j1,n3)=u2
50 continue
if (t2.lt.0) then
do 60 j2=1,4
u3=r(j2,3)
r(j2,3)=r(j2,1)
r(j2,1)=u3
60 continue
end if
do 70 n4=4, nst
v1=volum(r(1,2), r(1,3), r(1,n4))
if (v1.gt.e1) then
v2=volum(r(1,3), r(1,1), r(1,n4))
if (v2.gt.e1) then
v3=volum(r(1,1), r(1,2), r(1,n4))
if (v3.gt.e1) goto 1040
end if
end if
70 continue
print ' ', '----> Initial data are wrong <----'
print ' ', '----> stop <----'
stop 25
1040 continue
do 80 j2=1,4
u3=r(j2,4)
r(j2,4)=r(j2,n4)
r(j2,n4)=u3
80 continue
call circle(1)
call circle(2)
call circle(3)

```



```

call circle(4)
return
end

subroutine bar(trl,st,mk)

to-ka 'st' pone)setcq b tp-k 'trl' ppl pomoli
bapicentpi-eczego podpadlehiq

parameter(maxst=70000, maxtr=2*maxst)
integer next,sites,np,trl,tr2,tr3,tra,trb,st1,st2,st3
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
save

if(mk.ne.0) then
n0=next(mk,trl)
m0=mark(n0,trl)
endif
tr2=next(2,trl)
tr3=next(3,trl)
m2=mark(tr2,trl)
m3=mark(tr3,trl)
'tra' i 'trb' - dba hobyx tpeugolxhika
tra=ntr+1
trb=ntr+2
ntr=ntr+2
next(m2,tr2)=tra
next(m3,tr3)=trb
next(2,trl)=tra
next(3,trl)=trb
next(1,tra)=tr2
next(2,tra)=trb
next(3,tra)=trl
next(1,trb)=tr3
next(2,trb)=trl
next(3,trb)=tra
st1=sites(1,trl)
st2=sites(2,trl)
st3=sites(3,trl)
sites(1,trl)=st
sites(1,tra)=st
sites(2,tra)=st3
sites(3,tra)=st1
sites(1,trb)=st
sites(2,trb)=st1
sites(3,trb)=st2
if(mk.ne.0) then
n1=next(m0,n0)
call flip(n1,l)
call circle(n0)
endif
call circle(trl)
call circle(tra)
call circle(trb)
return
end

subroutine search(tgl,np,ok,mk)

poick tpeugolxhika b kotopyj popadaet to-ka np

parameter(maxst=70000, maxtr=2*maxst)
parameter (e-le=24
,e1-l-le=6)

integer next,sites,tgl,m,np,prev,ndp/0/
common /points/ nst,r(4,maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
logical ok
save

mk=0
a3=volum(r(1,np),r(1,sites(1,tgl)),r(1,sites(2,tgl)))
a1=volum(r(1,np),r(1,sites(2,tgl)),r(1,sites(3,tgl)))
a2=volum(r(1,np),r(1,sites(3,tgl)),r(1,sites(1,tgl)))
if(a1.ge.0.and.a2.ge.0.and.a3.ge.0) then
v1=abs(a1)
v2=abs(a2)
v3=abs(a3)
if(v1.le.e.or.v2.le.e.or.v3.le.e) then
if(v1.le.e.and.v2.le.e.and.v3.le.e) then
print *, ' all volumes are zero'
endif
if(v1.le.e) then
if(v2.le.e) then
ok=.false.
ndp=ndp+1
print *, ndp, '-th point is deleted'
if(scal(r(1,np),r(1,sites(3,tgl))).ge.e1) then
r(4,sites(3,tgl))=(r(4,np)+r(4,sites(3,tgl)))/2
endif
else
if(v3.le.e) then
if(scal(r(1,np),r(1,sites(2,tgl))).ge.e1) then
r(4,sites(2,tgl))=(r(4,np)+r(4,sites(2,tgl)))/2
endif
ok=.false.
ndp=ndp+1
print *, ndp, '-th point is deleted'
else
mk=1
ok=.true.
and if
endif
else
if(v2.gt.e) then
mk=3
ok=.true.
else
if(v3.le.e) then
if(scal(r(1,np),r(1,sites(1,tgl))).ge.e1) then
r(4,sites(1,tgl))=(r(4,np)+r(4,sites(1,tgl)))/2
endif
ok=.false.
ndp=ndp+1
print *, ndp, '-th point is deleted'
else
mk=2
ok=.true.
endif
endif
else
ok=.true.
and if
endif
endif
ok=.true.
end if
return

end if

```

4-11-71

adds.f

```

end if
if (nxt.ne.tg) goto 1010
return
end

logical function critter(tg,st)
  pprobepka kpugobogo kpteplq:
  popadaet li to-ka st b kohuc, opicahnyj bokpug tp-ka tg ?
  ecili popadaet, to critter = .false.

  parameter(maxst=70000, maxtr=2*maxst)
  integer next,sites,tg,st
  common /points/ nst,r(4,maxst)
  common /net/ ntr,next(3,maxtr),sites(3,maxtr)
  common /cones/ vec(3,maxtr)
  save

  if (scal(r(1,st),vec(1,tg)).gt.1) then
    critter=.false.
  else
    critter=.true.
  end if
  return
end

subroutine flip(tga,mal)

  parameter(maxst=70000, maxtr=2*maxst)
  integer next,sites,tga,tgb,shl
  common /points/ nst,r(4,maxst)
  common /net/ ntr,next(3,maxtr),sites(3,maxtr)
  save
  shl('m')=mod(m,3)+1

  ma2=shl(mal)
  ma3=shl(ma2)
  tgb=next(mal,tga)
  mb1=mark(tgb,tga)
  mb2=shl(mb1)
  mb3=shl(mb2)
  next(mark(next(ma2,tga),tga),next(ma2,tga))=tgb
  next(mark(next(mb2,tgb),tgb),next(mb2,tgb))=tga
  next(mal,tga)=next(mb2,tgb)
  next(mal,tgb)=next(mb3,tgb)
  next(ma2,tgb)=next(ma2,tga)
  next(ma2,tga)=tgb
  next(ma3,tgb)=tga
  sites(ma3,tga)=sites(mb1,tgb)
  sites(ma3,tgb)=sites(mb2,tgb)
  sites(mal,tgb)=sites(mal,tga)
  sites(ma2,tgb)=sites(ma3,tga)
  return
end

subroutine circle(tgi)

  by-icliq'tcq kompoehnty bektopa, happablehogo bdolx
  ocl opicahhogo bokpug tp-ka tgi kohuca i
  ine'lego taku' dilhu, -to ego ckalqphoe ppoizbedenie
  c bektopam! obparu'lix kohuca pabho l.

  parameter(maxst=70000, maxtr=2*maxst)

```

```

integer next,sites,tgi
common /points/ nst,r(4,maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
common /cones/ vec(3,maxtr)
real a(3,3),rhs(3),lw(3),w(6),x(3)
ind(1)=mod(1,3)+1
save

do 10 i1=1,3
  rhs(10)=1
  do 10 i1=1,3
    a(i1,i1)=r(i1),sites(10,tgi))
    call mal4a(3,3,0,a,3,rhs,x,lw,w,*1)
  do 20 j1=1,3
    vec(j1,tgi)=x(j1)
  return
end

real function scal(x,y)
  scal = (x,y)
  real x(3),y(3)

  scal=dprod(x(1),y(1))+dprod(x(2),y(2))+dprod(x(3),y(3))
  return
end

real function volum(x,y,z)
  ob'em co zhakom
  volume = (x,y,z)
  real x(3),y(3),z(3)
  real*8 vol

  vol=0
  do 10 j1=1,3
    m=mod(j1,3)+1
    n=mod(j1+1,3)+1
    vol=vol+x(j1)*(dprod(y(m),z(n))-dprod(y(n),z(m)))
  10 continue
  volum=vol
  return
end

integer function mark(tr1,tr2)
  integer tr1,tr2

  parameter(maxst=70000, maxtr=2*maxst)
  integer next,sites
  common /net/ ntr,next(3,maxtr),sites(3,maxtr)

  oppedelehie oplehtacil b tp-ke tr1 po tp-ku tr2
  do 10 mark=1,3
    if(next(mark,tr1).eq.tr2) return
  10 continue

  print *, ' ---> cpicok cmevhocetj he camocogiacobah <---'
  stop 51
end

subroutine output
  bybod dlq blizualizacil i intheppolqcll

  parameter(maxst=70000, maxtr=2*maxst)

```

adds.f

```

integer next, sites
common /points/ nst, r(4, maxst)
common /net/ ntr, next(3, maxtr), sites(3, maxtr)
logical addpts
save
addpts=.true.

write (7,*) ntr, nst
do 10 j2=1, nst
  write (7,*) ((r(i1, j2)*r(4, j2)), i1=1, 3)
  continue
10 do 11 j2=1, ntr
  write (7,*) sites(1, j2), sites(2, j2), sites(3, j2)
  continue
11 close(7)
if(.not. addpts) stop
write (2) ntr, nst
write (2) ((sites(i0, k0), i0=1, 3), k0=1, ntr)
write (2) ((next(i0, k0), i0=1, 3), k0=1, ntr)
close(2)
close(1)
return
end

```

```

subroutine output1
parameter(maxst=70000, maxtr=2*maxst)
integer next, sites
common /points/ nst, r(4, maxst)
common /net/ ntr, next(3, maxtr), sites(3, maxtr)
common /extremum/ rmin, rmax, radius, rad_ex
common /bld/ radv(maxtr), iord(maxtr)
logical addpts
logical frametr
save
addpts=.true.

ntr7 = 0

do 70 i = 1, ntr
  if(frametr(i, r, sites)) goto 70
  ntr7 = ntr7 + 1
  iord(ntr7) = i
70 continue
  write (7,*) ntr7, nst
  do 71 j2=1, nst
    write (7,*) ((r(i1, j2)*r(4, j2)), i1=1, 3)
    continue
  do 72 i3=1, ntr7
    j2 = iord(i3)
    write (7,*) sites(1, j2), sites(2, j2), sites(3, j2)
    continue
  close(7)
  if(.not. addpts) stop
  write (2) ntr, nst
  write (2) ((sites(i0, k0), i0=1, 3), k0=1, ntr)
  write (2) ((next(i0, k0), i0=1, 3), k0=1, ntr)
  close(2)
  close(1)
  return
end

```

addual.f

1

```

subroutine add(msize)
*
* adding of dual points
*
parameter (maxst=70000
,mmaxtr=2*maxst)
common /points/ nst,ntr
real r(3,maxst)
integer next(3,mmaxtr),sites(3,mmaxtr)
common /lala/next,sites,r
real radius(mmaxtr),center(2,mmaxtr)
real radv(mmaxtr),lord(mmaxtr)
common /work1/ radius,center, radv,lord
common /loder/ lfo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
,mdst,rdmax0,ldra,lwin,ldrm,msav,liq,miner,kva
,lpik,lpoq

open(3,file='addual',status='unknown',form='unformatted')
open(8,file='text')
call glider
write(8,*)'40 30'
write(8,*)' adding of dual points:'
write(8,15) 'the number of the points before adding is ',nst
15 format(a,i7)
* adding of dual points
if(mdst.gt.maxst) then
print *, '----> dimension maxst is too small <----'
print *, '----> put maxst = ',mdst, ' <----'
stop
end if
mdst1 = nst*msize
call addual(mdst1,rdmax,r,next,sites,radius,center,radv,lord)
write(8,15) 'the number of the points after adding is ',nst

ntr7=0
do 3777 ntr1=1,ntr
c if(radius(ntr1).ge.rdmx) goto 3777
ntr7=ntr7+1
lord(ntr7)=ntr1
3777 continue
write(3) ntr7,nst
write(3) ((r(i2,i3),i2=1,3),i3=1,nst)
write(3) ((sites(i2,int(lord(i3))),i2=1,3),i3=1,ntr7)
call plide(' addual ')
print *,% ---->,ntr7,'triangles <---'
ldr1=ldra
if(ldra.lt.0) then
print *, 'That is the trap'
print *, 'draw the new triangulation?'
call getans(ldr1)
endif
* if(ldr1.ne.0) call dtr1(150,r,sites)
c print *,% ----> the job is done <----'
close(3)
write(8,*) ' Hit right mouse button to continue'
close(8)
return
end

subroutine addual(mdst,rdmax,
,r,next,sites,radius,center,radv,lord)
parameter(rvmin=1e-8,rvmax=0.7)
common /points/ nst,ntr

```

```

real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
real radv(1)
integer lord(1)
integer tgl,temp
logical ok,frame,bad,border
*
if(mdst.le.nst) return
2000 continue
rav=rvmin
rma=rav
nst0=nst
nc=0
ral=0d0
naver=0
aver=0
tgl=1
do 789 i=1,ntr
if(bad(i,rav,radius,circle)) goto 789
r(1,nst+1)=center(1,i)
r(2,nst+1)=center(2,i)
r(3,nst+1)=0
call search(tgl,nst+1,ok,mk,r,next,sites)
if(.not.ok) goto 789
naver=naver+1
aver=aver+radius(i)
ral=amax(ral,radius(i))
789 continue
if(naver.eq.0) then
print *, 'can not add points'
write(8,*) 'can not add points'
return
endif
rma=aver/(1.01*naver)
continue
do 24 i=1,ntr
if(bad(i,rma,radius,circle)) goto 24
nss=1*nc+nst
r(1,nss)=center(1,i)
r(2,nss)=center(2,i)
r(3,nss)=0
call search(tgl,nss,ok,mk,r,next,sites)
if(.not.ok) goto 24
if(border(tgl,r,sites,radius)) goto 24
do 644 k=1,3
nt=next(k,1)
if(nt.gt.i) goto 644
if(bad(nt,rma,radius,circle)) goto 644
r(1,nss+1)=center(1,nt)
r(2,nss+1)=center(2,nt)
ngl=tgl
call search(ngl,nss+1,ok,mk,r,next,sites)
if(.not.ok) goto 644
if(border(ngl,r,sites,radius)) goto 644
ra=(center(1,nt)-center(1,i))*2
ra=(center(2,nt)-center(2,i))*2+ra
if(ra.lt.rav) goto 24
644 continue

```

```

nc=nc+1
if(nss.ge madst) goto 103
24 continue
103 continue
c      if(nc.le.1) then
      print *, ' can not add more points', rma, rav
      write(8,'') can not add more points'
      return
      endif
nss=nc+nst

do 890 ms=nst0+1,nss
call search(tgl,ms,ok,mk,r,next,sites)
  if(.not.ok) goto 890
  nst=nst+1
  r(1,nst)=r(1,ms)
  r(2,nst)=r(2,ms)
  call bar(tgl,nst,mk,r,next,sites,radius,center)
  call recon(tgl,r,next,sites,radius,center)
  continue
890 print *, ' --->',nst-nst0,' out of ',madst-nst0,
c      ' dual points have been added <---'
  if(madst.gt.nst) goto 2000

c1070 print *,
c  & ' ---> the addition of dual points is completed <---'
c      print *, ' ---> # pnt =',nst,' <=== '
      return
      end

```

graphics.f

1

```

subroutine graphics(lgld)
#include<fgl.h>
#include<device.h>
parameter(maxst=70000, maxtr=2*maxst)
integer next, sites
common /points/ nost, r(4, maxst)
common /net/ ntr, next(3, maxtr), sites(3, maxtr)
common/mode/ exitmode, pictmode, pointmode
common/extremum/rmin, rmax, radius, rad_ex
integer exitmode, pictmode, pointmode

call init(lgld)
call clear
call zclear
call swapbu
do while(1)
do while(qtest(0).eq.0)
call Draw
end do
call process
if(exitmode .eq.0) goto 5
end do
return
5
end

subroutine init(lgld)
#include<fgl.h>
#include<device.h>
parameter(NORMAL = 1, HOLE = 0)
parameter(w = 0.45, h = 0.1, x1 = 1.6, y1 = -0.6, dist = 0.18)
common/mode/ exitmode, pictmode, pointmode
common/extremum/rmin, rmax, radius, rad_ex
common/shtri/ sh(4), co(3), con(3)
common/rots/ angl(7), pov(3)
common/col/ color, mcolor, legend(3, 13)
common/popup_stuff/menu
integer exitmode, pictmode, pointmode
real backvec(3)/0.0, 0.0, 0.0/
call prefpo(0, 1270, 0, 1000)
call foregr
lgld = winope("geosphere", 13)
call RGBmod
call double
call gconfl
call lsetde(0, 57ffff)
call zbuffe(.TRUE.)
call qdevic(RIGHTM)
call qdevic(LEFTMO)
call tie(RIGHTM, MOUSEX, MOUSEY)
call tie(LEFTMO, MOUSEX, MOUSEY)
menu = newpup()
call addtop(menu, 'DATA CHANGE $TIWIRED MODEL$MAP$SMOOTH MAP', 50, 0)
call addtop(menu, ' add point$remove point$EXIT ', 50, 0)
exitmode = 1
pictmode = 1
pointmode = NORMAL
do 12 1 = 2, 4
sh(1) = x1*0.001
do 16 1 = 5, 7
angl(1) = x1
call c3f(backvec)
color = x1*w

```

```

mcolor = 12
rad_ex = x1*w/8 + rmin*w*0.75
return
end

subroutine process
#include<fgl.h>
#include<device.h>
common/mode/ exitmode, pictmode, pointmode
common/popup_stuff/menu
common/extremum/rmin, rmax, radius, rad_ex
integer exitmode, pictmode, pointmode
integer*2 val, mx, my
integer dev
integer menuval, exitmode, pictmode, pointmode
dev = qtest()
do 1000 while (dev .ne. 0)
dev = qread(val)
if(dev .eq. LEFTMO) then
if(val .eq. 1) then
l = qread(mx)
k = qread(my)
call move_shtri(h(mx, my))
call Draw_shtri(h(mx, my))
do 1001 while (qtest() .eq. 0)
call Draw
1001 end do
end if
else
if(dev .eq. RIGHTM) then
menuval = dopup(menu)
if(menuval .eq. 1) then
pictmode = 1
call Draw
end if
if(menuval .eq. 2) then
pictmode = 2
call Draw
end if
if(menuval .eq. 3) then
pictmode = 3
call Draw
end if
if(menuval .eq. 4) then
call add_point
call Draw
end if
if(menuval .eq. 5) then
call remove_point
call Draw
end if
if(menuval .eq. 6) then
exitmode = 0
if(exitmode .eq. 0) goto 6
end if
end if
end if
end if
1000 end do
6
return
end

```

```

subroutine Draw
#include<fgl.h>
#include<device.h>
common/shtri/ sh(4), co(3), con(3)
common/rots/ angl(7), pov(3)
common/col/ color, mcolor, legend(3,13)
common/extremum/rmin, rmax, radius, rad_ex
character string1*5, string2*5, string3*5, string4*5
real backvec(3)/0.0,0.0,0.0/
real greenvec(3)/1.0,1.0,1.0/

real ldmatrix(4,4)/
4 0.5,0.0,0.0,0.0,
4 0.0,0.5,0.0,0.0,
4 0.0,0.0,0.5,0.0,
4 0.0,0.0,0.0,0.5/

real kvadrat1(3)/-1.0,-1.0,-1.0/
real kvadrat2(3)/-1.0,1.0,-1.0/
real kvadrat3(3)/1.0,1.0,-1.0/
real kvadrat4(3)/1.0,-1.0,-1.0/

write(string1,'X')
write(string2,'Y')
write(string3,'Z')
write(string4,'O')
call c3f(backvec)
call clear
call zclear

call ortho(-1.5,2.1,-1.5,1.5,-1.5,1.5)
call loadma(ldmat)
call palette(ldmat)
call pushma
call rot(pov(5),'x')
call rot(pov(6),'y')
call rot(pov(7),'z')
call c3f(greenvec)
call cmov(1.03,-1.03,-1.03)
call charst(string1,5)
call cmov(-1.03,1.03,-1.03)
call charst(string2,5)
call cmov(-1.03,-1.03,1.03)
call charst(string3,5)
call cmov(-1.03,-1.03,-1.03)
call charst(string4,5)
call c3f(greenvec)
call bgnclo
call v3f(kvadrat1)
call v3f(kvadrat2)
call v3f(kvadrat3)
call v3f(kvadrat4)
call endclo

call pushma
call transl(0.0,0.0,2.0)
call c3f(greenvec)
call bgnclo
call v3f(kvadrat1)

```

```

call v3f(kvadrat2)
call v3f(kvadrat3)
call v3f(kvadrat4)
call endclo
call popmat
call pushma
call rotate(-900,'y')
call c3f(greenvec)
call bgnclo
call v3f(kvadrat1)
call v3f(kvadrat2)
call v3f(kvadrat3)
call v3f(kvadrat4)
call endclo
call popmat
call pushma
call transl(-2.0,0.0,0.0)
call rotate(-900,'y')
call c3f(greenvec)
call bgnclo
call v3f(kvadrat1)
call v3f(kvadrat2)
call v3f(kvadrat3)
call v3f(kvadrat4)
call endclo
call popmat
call Obj
call make_metka
call popmat
call swapbu
return
end

subroutine palette(ldmat)
#include<fgl.h>
#include<device.h>
common/shtri/ sh(4), co(3), con(3)
common/rots/ angl(7), pov(3)
common/col/ color, mcolor, legend(3,13)
common/extremum/rmin, rmax, radius, rad_ex
parameter(w = 0.45, h = 0.1, xl = 1.6, yl = -0.6, dist = 0.18)
parameter(dl = 0.005)
real legend(3,13)/
40.0,0.0,0.8,
40.0,0.0,1.0,
40.0,0.6,0.6,
40.0,0.8,0.0,
40.0,1.0,0.0,
40.2,0.9,0.0,
41.0,1.0,0.0,
41.0,0.8,0.0,
41.0,0.6,0.0,
41.0,0.4,0.0,
41.0,0.2,0.0,
41.0,0.0,0.0,
40.0,0.0,0.0/

real bluevec(3)/0.0,0.8,1.0/
real redvec(3)/1.0,0.0,0.0/
real bluevec1(3)/0.0,0.0,1.0/
real greyvec(3)/0.8,0.8,0.8/
real blackvec(3)/0.0,0.0,0.0/

```



```

character*40 string(8)
write(string(1),'rmin radius rmax'
write(string(2),'0 X coordinate 1'
write(string(3),'0 Y coordinate 1'
write(string(4),'0 2 coordinate 1'
write(string(5),'0 X rotation 360'
write(string(6),'0 Y rotation 360'
write(string(7),'0 2 rotation 360'
write(string(8),'Hole rmin rmax 1.5'
call c3f(blackvec)
call rectf(1.5,-1.5,2.1,1.5)
call c3f(greyvec)
call rectf(1.5,-1.5,2.1,1.5)
do 20 i = 1,7
x = x1
y = dist*i + h*(i-1)+y1
call c3f(blackvec)
call rectf(x-dl,y-dl,x+w*dl,y+h*dl)
call c3f(bluevec)
call rectf(x,y,x+w,y+h)
call c3f(blackvec)
call cmov(x1,y-0.04,0.1)
call charst(string(1),30)
20 continue
do 21 k = 2,7
y2= dist*k + h*(k-1)+y1
do 23 j = 1,11
if(mod(j,6) .eq.0)then
h1 = 0.03
else
h1 = 0.02
if(mod(j,3) .eq.0)then
h1 = 0.01
else
h1 = 0.002
x2= x1+w/12.*j
call c3f(blackvec)
call rectf(x2,y2,x2+w1,y2+h1)
23 continue
21 continue
do 22 j = 0,11
y = dist +y1
x = x1+w/12.*j
call c3f(legend(1,j+1))
call rectf(x,y,x+w/12.,y+h)
22 continue
y = y1
x = x1
w1 = 0.002
h1 = 0.03
w2 = 0.004
call c3f(blackvec)
call rectf(x-dl,y-h-dl,x+w*dl,y+dl)
call c3f(bluevec)
call rectf(x,y-h,x+w,y)
call c3f(blackvec)
x = x1
call rectf(x,y-h,x+w/8,y)
call c3f(blackvec)
call rectf(x,y-h,x+w2,y-h+h1)
x5 = x1 + w/8 +rmin*w*0.875/1.5

call c3f(bluevec1)
call rectf(x5,y-h,x5+w1,y)
x5 = x1 + w/8+rmax*w*0.875/1.5
call c3f(redvec)
call rectf(x5,y-h,x5+w1,y)
call c3f(blackvec)
call cmov(x1-0.07,y-h-0.04,0.1)
call charst(string(8),30)
call shtrih
return
end

subroutine shtrih
common/shtri/ sh(4),co(3),con(3)
common/rots/ angl(7),pov(3)
common/col/ color,mcolor,legend(3,13)
common/extremum/rmin,rmax,radius,rad_ex
parameter(w = 0.45, h = 0.1,x1 = 1.6,y1 = -0.6,dist = 0.18)
parameter(del = 0.005)
real blackvec(3)/0.0,0.0,0.0/
i = 0
y0 = y1-h
call c3f(blackvec)
call rectf(rad_ex,y0,rad_ex+del,y0+h)
i = i + 1
y0 = y1 + dist*i + h*(i-1)
call c3f(blackvec)
call rectf(color,y0,color+del,y0+h)
do 11 i = 2,4
y0 = y1 + dist*i + h*(i-1)
call c3f(blackvec)
call rectf(sh(i),y0,sn(i)+del,y0+h)
11 continue
do 15 i = 5,7
y0 = y1 + dist*i + h*(i-1)
call c3f(blackvec)
call rectf(angl(i),y0,angl(i)+del,y0+h)
15 continue
return
end

subroutine move_shtrih(mx,my)
parameter(NORMAL = 1,HOLE = 0)
parameter(w = 0.45, h = 0.1,x1 = 1.6,y1 = -0.6,dist = 0.18)
parameter(del = 0.005)
common/shtri/ sh(4),co(3),con(3)
common/rots/ angl(7),pov(3)
common/col/ color,mcolor,legend(3,13)
common/mode/exitmode,pictmode,pointmode
common/extremum/rmin,rmax,radius,rad_ex
integer*2 mx,my
integer exitmode,pictmode,pointmode
real whitevec(3)/1.0,1.0,1.0/
real x3(13),x4(6)

Metky na shkale radiusov
x4(1) = x1
x4(2) = x1+w/8
x4(3) = x4(2) + rmin * 0.875 *w/1.5
x4(4) = x4(2) + rmax * 0.8888888875 *w/1.5
x4(5) = x4(2) + 0.875* w/1.5
x4(6) = x1+w

```

graphics.f

4

```

x = (3.6/1270)*float(mx) - 1.5
y = (3.0/1000)*float(my) - 1.5

if (y.le.(yl-h).or. x.lt.xl .or. x.gt. xl+w ) then
  n = -1
else
  n = int((y - yl)/(dist+h))+1
  if (y - (yl + dist*n + h*(n-1))) .le.0) n = -1
  if (y .le. yl .and. y .gt. yl-h) n=0
  if (n.eq.0) then
    do 20 i = 1,6
      if ((abs(x-x4(i))) .le.2*del) then
        x = x4(i)
        goto 19
      end if
    continue
  20 rad_ex = x
  call make_change(x4)
end if
if (n.eq.1) then
  color = x
  call make_change_back
end if
if (n.le.4.and.n.ge.2) then
  sh(n) = x
end if
if (n.ge.5 .and.n.le.7) then
  do 17 i = 0,12
    x3(i) = x1*(w/12)*i
    if ((abs(x-x3(i))) .le.del) then
      x = x3(i)
      goto 18
    end if
  17 continue
  18 angl(n) = x
end if
end if
do 14 l = 5,7
  pov(l) = (angl(l) - x1)/w)*360
  14 continue
  if (pointmode .eq.NORMAL) then
    radius = ((color - x1)/w)*(rmax-rmin)+rmin
    mcolor = ((color - x1)/w)*11.*1
    end if
    co(1) = ((sh(2) - x1)/w)*2. - 1.
    co(2) = ((sh(3) - x1)/w)*2. - 1.
    co(3) = ((sh(4) - x1)/w)*2. - 1.
    x = co(1)
    y = co(2)
    z = co(3)
    .....calc of the position of the normal.vec *
    s = sqrt( 1.0/(1.e-10*x**2+y**2+z**2))
    con(1) = s*x
    con(2) = s*y
    con(3) = s*z
    .....calc of the position of the point on the beam*
    co(1) = con(1)*radius
    co(2) = con(2)*radius
    co(3) = con(3)*radius

return
end

subroutine make_metka
#include<gl.h>
#include<device.h>
parameter(NORMAL = 1,HOLE = 0)
parameter(w = 0.45, h = 0.1,x1 = 1.6,y1 = -0.6,dist = 0.18)
parameter(piram=.015)
common/shtri/ sh(4),co(3),con(3)
common/mode/exitmode,pictmode,pointmode
common/rots/ angl(7),pov(3)
common/co/ color,mcolor,legend(3,13)
common/extremum/rmin,rmax,radius,rad_ex
integer exitmode,pictmode,pointmode
real metka1(3),metka2(3),metka3(3),metka4(3),metka5(3)
real metka6(3),metka7(3)
real beam(3)
integer*4 n
real greenvec(1)/1.0,1.0,1.0/
real zero(3)/0.0,0.0,0.0/
.....calc of the position of the beam
radius = ((color - x1)/w)*(rmax-rmin)+rmin
s1 = sqrt( 1.0/(1.e-10*x**2+y**2+z**2))
beam(1) = s1*con(1)
beam(2) = s1*con(2)
beam(3) = s1*con(3)
.....calc of the metka on the beam*
metka1(1) = -con(2)*piram*radius*con(1)
metka1(2) = con(1)*piram*radius*con(2)
metka1(3) = radius*con(3)
metka2(1) = con(2)*piram*radius*con(1)
metka2(2) = -con(1)*piram*radius*con(2)
metka2(3) = radius*con(3)
metka3(1) = radius*con(1)
metka3(2) = -con(3)*piram*radius*con(2)
metka3(3) = con(2)*piram*radius*con(3)
metka4(1) = radius*con(1)
metka4(2) = con(3)*piram*radius*con(2)
metka4(3) = -con(2)*piram*radius*con(3)
metka5(1) = -con(3)*piram*radius*con(1)
metka5(2) = radius*con(2)
metka5(3) = con(1)*piram*radius*con(3)
metka6(1) = con(3)*piram*radius*con(1)
metka6(2) = radius*con(2)
metka6(3) = -con(1)*piram*radius*con(3)
metka7(1) = (4*piram*radius)*con(1)
metka7(2) = (4*piram*radius)*con(2)
metka7(3) = (4*piram*radius)*con(3)
if (pointmode .eq. NORMAL) then
  ..... metka*
  call c3f(greenvec)
  call bgnpol
  call v3f(metka1)
  call v3f(metka2)
  call v3f(metka7)
  call v3f(metka1)
  call endpol
  call bgnpol
  call v3f(metka3)
  call v3f(metka4)

```

```

call v3f(metka7)
call v3f(metka3)
call endpol
call bgmpol
call v3f(metka5)
call v3f(metka6)
call v3f(metka7)
call v3f(metka5)
call endpol
end if
..... beam
call c3f(greenvec)
call bgmlin
call v3f(zero)
call v3f(beam)
call endlin
return
end

subroutine Obj
#include<gl.h>
#include<device.h>
parameter(maxst=70000, maxtr=2*maxst)
common/shr1/ sh(4),co(3),con(3)
common/rots/ angl(7),pov(3)
common/col/ color,mcolor,legend(3,13)
common/extremum/rmin,rmax,radius,rad_ex
integer next,sites
integer pictmode,pointmode
real bordovec(3)/0.0,0.0,0.0/
common /points/ nst,r(4,maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
common/mode/exitmode,pictmode,pointmode
real idnat(4,4)
real x(3),y1(3),y2(3)
integer z1
..... MAP MODEL .....
if(pictmode.eq.2) then
do 30 l = 1,ntr
z1 = 1
do 31 k = 1,3
if(r(4,sites(k,1)).gt.1.5) then
z1 = 0
y1(1) = r(1,sites(sh(k+1,1))*r(4,sites(sh(k+1,1)))
y1(2) = r(2,sites(sh(k+1,1))*r(4,sites(sh(k+1,1)))
y1(3) = r(3,sites(sh(k+1,1))*r(4,sites(sh(k+1,1)))
y2(1) = r(1,sites(sh(k+2,1))*r(4,sites(sh(k+2,1)))
y2(2) = r(2,sites(sh(k+2,1))*r(4,sites(sh(k+2,1)))
y2(3) = r(3,sites(sh(k+2,1))*r(4,sites(sh(k+2,1)))
end if
31 continue
if(z1.eq.1) then
r_av = (r(4,sites(1,1))+r(4,sites(2,1))+r(4,sites(3,1)))/3.
mcol = 12.*(r_av - rmin)/(rmax-rmin)+1
call c3f(legend(i,mcol))
call bgmpol
do 32 j = 1,3
x(1) = r(1,sites(j,1))*r(4,sites(j,1))
x(2) = r(2,sites(j,1))*r(4,sites(j,1))
x(3) = r(3,sites(j,1))*r(4,sites(j,1))
call v3f(x)
32 continue
call endpol
end if
else
call c3f(bordovec)
call bgmpol
do 932 j = 1,3
r_av=r(4,sites(j,1))
if(r_av.ge.1.5) then
x(1)=0
x(2)=0
x(3)=0
else
x(1) = r(1,sites(j,1))*r_av
x(2) = r(2,sites(j,1))*r_av
x(3) = r(3,sites(j,1))*r_av
endif
end if
call v3f(x)
932 continue
call endpol
end if
end if
30 continue
end if
..... WIRED MODEL .....
if(pictmode.eq.1) then
do 40 l = 1,ntr
z1 = 1
do 41 k = 1,3
if(r(4,sites(k,1)).gt.1.5)then
z1 = 0
end if
41 continue
if(z1.eq.1)then
call bgnclo
do 42 j = 1,3
mcol = 12.*(r(4,sites(j,1)) - rmin)/(rmax-rmin)+1
x(1) = r(1,sites(j,1))*r(4,sites(j,1))
x(2) = r(2,sites(j,1))*r(4,sites(j,1))
x(3) = r(3,sites(j,1))*r(4,sites(j,1))
call c3f(legend(i,mcol))
call v3f(x)
42 continue
call endclo
and if
40 continue
end if
..... SMOOTH MAP MODEL .....
if(pictmode.eq.3) then
do 50 l = 1,ntr
z1 = 1
do 51 k = 1,3
if(r(4,sites(k,1)).gt.1.5)then
z1 = 0
end if
51 continue
if(z1.eq.1)then
call bgmpol
do 52 j = 1,3
mcol = 12.*(r(4,sites(j,1)) - rmin)/(rmax-rmin)+1
if(mcol.lt.1)mcol = 1
if(mcol.gt.12)mcol = 12
x(1) = r(1,sites(j,1))*r(4,sites(j,1))
x(2) = r(2,sites(j,1))*r(4,sites(j,1))

```

graphics.f

6

```

x(3) = r(3,sites(j,1))*r(4,sites(j,1))
call c3f(legend(1,mcol))
call v3f(x)
52 continue
call endpol
else
call c3f(bordovec)
call bgnpol
do 952 j = 1,3
r_av=r(4,sites(j,1))
if(r_av.ge.1.5) then
x(1)=0
x(2)=0
x(3)=0
else
x(1) = r(1,sites(j,1))*r_av
x(2) = r(2,sites(j,1))*r_av
x(3) = r(3,sites(j,1))*r_av
endif
952 ntinue
call endpol

end if
50 continue
end if
return
end

subroutine add_point
parameter (NORMAL = 1, HOLE = 0)
parameter (maxst=70000, maxtr=2*maxst)
parameter (w = 0.45, h = 0.1,x1 = 1.6,y1 = -0.6,dist = 0.18)
common /points/ nst,r(4,maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
common /mode/ exitmode,pictmode,pointmode
common /shtr1/ sh(4),co(3),con(3)
common /rots/ angl(7),pov(3)
common /col/ color,mcolor,legend(3,13)
common /extremum/rmin,rmax,radius,rad_ex
integer exitmode,pictmode,pointmode
integer next,sites,tgl
logical ok
radius = ((color - x1)/w)*(rmax-rmin)+rmin
r(1,nst+1) = con(1)*radius
r(2,nst+1) = con(2)*radius
r(3,nst+1) = con(3)*radius
r(4,nst+1) = radius
rad = plram
tgl = 1

call search(tgl,nst+1,ok,mk)
call distance(rad,m,nst+1)
n = sites(m,tgl)
do 70 i = n,nst+1
do 71 k = 1,4
r(k,i) = r(k,i+1)
70 continue

nst = nst+1
tgl = 1
call tetra

do 73 np = 5,nst
call search(tgl,np,ok,mk)
if(ok)then
call bar(tgl,np,mk)
call recon(tgl)
end if
73 continue
mbe1 = 1
call setbel(mbe1)
call ringbe
return
end

subroutine distance(rad,kmin,np)
parameter (maxst=70000, maxtr=2*maxst)
parameter (w = 0.45, h = 0.1,x1 = 1.6,y1 = -0.6,dist = 0.18)
common /points/ nst,r(4,maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
common /mode/ exitmode,pictmode,pointmode
common /shtr1/ sh(4),co(3),con(3)
common /rots/ angl(7),pov(3)
common /col/ color,mcolor,legend(3,13)
common /extremum/rmin,rmax,radius,rad_ex
integer exitmode,pictmode,pointmode
integer next,sites,tgl
logical ok
real x(3),y(3)
radius = ((color - x1)/w)*(rmax-rmin)+rmin
dist1 = 0

```

graphics.f

7

```

do 80 k = 1, 3
do 81 j = 1, 3
x(j) = r(j, np) * r(4, np)
y(j) = r(j, sites(k, tgl)) * r(4, sites(k, tgl))
dist1 = dist1 + (x(j) - y(j)) ** 2
81 continue
if (k .eq. 1) then
dmin = dist1
kmin = 1
else if (dist1 .lt. dmin) then
kmin = k
dmin = dist1
end if
80 continue
if (dist1 .lt. rad) m = -1
return
end

```

```

integer pointmode
pointmode = NORMAL
radius = ((color - x1) / w) * (rmax - rmin) + rmin
rad_ex = radius * 0.875 * w / 1.5 + x1 * w / 8
return
end

function sh(k)
sh = mod(k+2, 3) + 1
return
end

```

```

subroutine make_change(x4)
parameter(NORMAL = 1, HOLE = 0)
parameter(maxst=70000, maxtr=2*maxst)
parameter(w = 0.45, h = 0.1, x1 = 1.6, y1 = -0.6, dist = 0.18)
parameter(del = 0.015)
common/col/ color, mcolor, legend(3, 13)
common/extremum/rmin, rmax, radius, rad_ex
common/mode/exitmode, pictmode, pointmode
integer pointmode
real x4(6)
if (rad_ex .ge. x4(1) .and. rad_ex .lt. x4(2) * del) then
color = x1
radius = 10.0
mcolor = 13
pointmode = HOLE
end if
if (pointmode .eq. NORMAL) then
if (rad_ex .le. x4(3) .and. rad_ex .ge. x4(2) * 2 * del) then
rmin = (rad_ex - x1 - w / 8) * 1.5 / (0.875 * w)
if (rmin .le. del) rmin = 0
color = x1
end if
if (rad_ex .gt. x4(4)) then
rmax = (rad_ex - x1 - w / 8) * 1.5 / (0.875 * w)
color = x1 * w
end if
if (rad_ex .lt. x4(4) .and. rad_ex .gt. x4(3)) then
color = ((rad_ex - x4(3)) * w / (x4(4) - x4(3))) * x1
end if
end if
return
end

```

```

subroutine make_change_back
parameter(maxst=70000, maxtr=2*maxst)
parameter(w = 0.45, h = 0.1, x1 = 1.6, y1 = -0.6, dist = 0.18)
parameter(NORMAL = 1, HOLE = 0)
common/col/ color, mcolor, legend(3, 13)
common/mode/exitmode, pictmode, pointmode
common/extremum/rmin, rmax, radius, rad_ex

```

```

subroutine int(i)
  interpolation
  'nst' = 0 points, 'r' - their coordinates
  'ntr' = 0 triangles
  radcir = radius*2 of circle for each triangle
  parameter(maxst=70000
    ,minst= 70000
    ,maxtr=2*minst)
  parameter(nbndp=11)
  parameter(nfunc=5)
  common /points/ nst,ntr
  real r(3,maxst)
  integer next(3,maxtr),sites(3,maxtr)
  common /ala/r,next,sites
  real tang(nfunc,minst)
  real radcir(maxtr),wight(minst)
  common /work1/ radcir,wight,tang
  common /llder/ lfo,lsu,xm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
    madst,rdmax0,ldra,twin,ldrm,msav,lig,miner,kva
    ,lplk,lpog
  parameter(maxlv=20)
  integer work(6*maxlv+9*maxst)
  open(1,file='tempor',status='unknown',form='unformatted')
  open(2,file='actual',status='unknown',form='unformatted')
  open(7,file='triang')
  open(9,file='rescales',status='unknown',form='unformatted')
  open(8,file='text3')
  call qllder
  read (2,end=1010,err=1010) ntr,nst1
  read (2,end=1010,err=1010) ((r(i0,i1),i0=1,3),i1=1,nst1)
  print *, '====',nst1,' points have been read'
  format(a,17,a)
  if(nst1.gt.maxst) then
    print *, '====> dimension maxst is too small <===='
    print *, '====> put maxst = ',nst1,' <===='
    stop 18
  end if
  goto 1020
  write(8,*) '40 30'
  write(8,*) ' interpolation is not aloud'
  write(8,*) 'The second input parameter is too small'
  close(8)
  call try1('text3')
  return
1020 continue
  write(8,*) '40 30'
  write(8,*) ' interpolation is done '
  write(8,16) ' ',nst1,' points have been read'
  read (1,end=1010,err=1010) ntr,nst
  read (1,end=1010,err=1010) ((sites(i4,i5),i4=1,3),i5=1,ntr)
  read (1,end=1010,err=1010) ((next(i2,i3),i2=1,3),i3=1,ntr)
  print *, '====',ntr,' triangles have been read <===='
  if(nst.gt.minst) then
    print *, '====> dimension minst is too small <===='
    print *, '====> put minst = ',nst,' <===='
    stop 13
  end if
  do 7659 lfp=1,nst-nbndp
    read(9,end=1010,err=1010) dum1,dum2,dum3,wight(lfp)
    continue
    errors have been read <===='
  end if
  ismt=kva
  lkill=0
  if(lplk.ge.1) lkill=1
  lstat=0
  if(lplk.eq.1.and.ismt.eq.1) lstat=1
  .....
  * ismt= 0 - no smoothing
  * ismt= 1 - errors are taken from the file with c r(...)
  * ismt= 2 - errors are equal 1
  * lkill= 0 - don't kill the peaks
  * lkill= 1 - kill the peaks
  * lstat=1 - take into account errors while killing peaks
  * lstat=0 - don't take into account errors while killing p.
  .....
  if(lplk.lt.0) then
    print *, ' kill the peaks ?'
    call getans(lplk)
  if(lplk.ne.0) then
    lkill=1
    if(lpog.ne.0) then
      print *, ' take into account errors ?'
      call getans(lpog)
      if(lpog.ne.0) lstat=1
    endif
  end if
  end if
  if(ismt.lt.0) then
    print *, ' make smoothing ?'
    call getans(11)
    if(11.ne.0) then
      ismt=2
      if(lpog.ne.0) then
        print *, ' take into account errors ?'
        call getans(lpog)
        if(lpog.ne.0) ismt=1
      endif
    endif
  end if
  if(lkill.eq.1) then
    call smstat(lstat,r,wight,next,sites,minst,work)
  end if
  if(ismt.ge.1) call smooth(r,wight,next,sites,work,minst,ismt)
  call preint(r,next,sites,radcir,tang,work,minst)
  call interp(ntr,nst),r,next,sites,tang,work,maxlv,maxst)
  print *, '====> write on the disk <===='
  call outint(ntr,nst1,r,work)
  close(7)
  call pllder('interp')

```

```

c
write(0,*)' Hit right mouse button to continue'
close(0)
if (ind.eq.1) then
call try1('text3')
endif
print *, '==== end of job <===='
close(1)
close(2)
close(3)
close(9)
return
end

subroutine interp(ntrl,nst1,r,next,sites,tang,w,maxlv,maxst)
integer w(6*maxlv+9*maxst)
real tang(1)
real r(1)
integer next(1),sites(1)
llo=1
lmo=llo+maxst*2
lib=lmo+maxst*2
lxl=lib+maxst*2
lor=lxl+maxst*2
lco=lor+maxst
lno=lco+maxlv*4
lli=llc+maxst
print *, 'lno=',lno
print *, 'total memory =',lno*maxlv*2
call prepar(ntrl,nst1,r,next,sites,w(lco),w(lno))
6,w(llo),w(lmo),w(lib),w(lxl),w(lor),w(lli),maxlv,maxst)

c
print *, '==== interpolation <===='
write(0,15) ' model of surface is constructed'
15 format(a)
call adppts(ntrl,nst1,r,sites,tang,w(llo),w(lli))
return
end

subroutine prepar(ntrl,nst,r,next,sites
6,corner,nn,lord,mord,ibf,x,order,1ist,maxlv,maxst)

na=alxhaq podgotobka
parameter(maxlv=20)
'nst' = 0 to ek, 'r' = 1x koopulhaty, 'w' = pogpe(hoctl.
common /points/ nst0,nst0
real r(3,maxst)
integer next(3,1),sites(3,1)
integer corner(4,maxlv),nn(2,maxlv),lord(maxst,2),mord(maxst,2)
integer ibf(maxst*2)
real x(2,maxst),order(maxst)
integer 1ist(0:1)
equivalenc (1ist(0),lord(1,2))

ii=1nt(log(real(maxst))/log(3.0))+1
if (11.gt.maxlv) then
print *, '==== change parameters. put maxlv= ',ii, '<===='
stop
endif
nd=nst-nst0
do 10 j0=1,nd
x(1,j0)=r(1,j0+nst0)
x(2,j0)=r(2,j0+nst0)
continue

c
parameter(nfunc=5)
integer st,sh
common /points/ nst,ntr
real r(3,1)
integer sites(3,1)
real tang(nfunc,1)
integer lord(maxst),1ist(0:maxtr)
integer lord(1),1ist(0:1)
real h2(3),a(4,3),c(6,3)
sh(m)=mod(m,3)+1
save
ns=1ist(0)
do 50 nt=1,ntr
nds=1ist(nt)-ns
if (nds.gt.0) then
nset=nt
do 40 k0=1,nds
np=lord(k0+ns)
call wghts(h2,nt,np,r,sites)
1htcappolqclq: bzbe(ethhe cpdhee ot tpex bep(1h

```

intim.f

3

```

*
*
      h2(1)+h2(2)+h2(3)=1
      z=0
      do 30 nv=1,3
        st=sites(nv,nt)
        w=funcal(st,np,r,tang)
        z=z+w*h2(nv)
        continue
      30   r(3,np)=z
      40   continue
      end if
      ns=1st(int)
      50   continue
      return
      end

*
*
      subroutine wghts(h,nt,np,r,sites)
*
*
      h - ploadi tpox obpazobablixq tpeugolxhkob.
      parameter (eps=1e-8)
      integer st1,st2,st3
      common /points/ nst,ntz
      real r(3,1)
      integer sites(3,1)
      real h(3)
      save

      st1=sites(1,nt)
      st2=sites(2,nt)
      st3=sites(3,nt)
      h1=abs(r(1,st2)-r(1,np))*r(2,st3)-r(2,np))
      h2=abs(r(1,st3)-r(1,np))*r(2,st1)-r(2,np))
      h3=abs(r(2,st3)-r(2,np))*r(1,st1)-r(1,np))
      r1=(r(1,st1)-r(1,np))*2+(r(2,st1)-r(2,np))*2
      r2=(r(1,st2)-r(1,np))*2+(r(2,st2)-r(2,np))*2
      r3=(r(1,st3)-r(1,np))*2+(r(2,st3)-r(2,np))*2
      if(r1.lt.eps) then
        h(1)=1
      else if(r2.lt.eps) then
        h(2)=0
      else if(r3.lt.eps) then
        h(3)=0
      else if(r2.lt.eps) then
        h(1)=0
      else if(r3.lt.eps) then
        h(1)=0
      else
        h(1)=1
        h(2)=0
        h(3)=0
      end if
      h(1)=h(1)/z
      h(2)=h(2)/z
      h(3)=h(3)/z
      return
      end

*
*
      h(1)=(h2**2+r2**2+h3**2+r3**2)*(h1)**2
      h(2)=(h3**2+r3**2+h1**2+r1**2)*(h2)**2
      h(3)=(h1**2+r1**2+h2**2+r2**2)*(h3)**2
      z=h(1)+h(2)+h(3)
      h(1)=h(1)/z
      h(2)=h(2)/z
      h(3)=h(3)/z
      return
      end

*
*
      subroutine searchint(tgl,np,r,next,sites)
*
*
      polick tpeugolxhika b kotopyj popadaet to-ka np
      integer tgl,m,np,prev
      common /points/ nst,ntr
      real r(3,1)
      integer next(3,1),sites(3,1)
      save

      mk=0
      a3=volum(r(1,np),r(1,sites(1,tgl)),r(1,sites(2,tgl)))
      a1=volum(r(1,np),r(1,sites(2,tgl)),r(1,sites(3,tgl)))
      a2=volum(r(1,np),r(1,sites(3,tgl)),r(1,sites(1,tgl)))
      if(a1.ge.0.and.a2.ge.0.and.a3.ge.0) then
        return
      end if
      if(a2.lt.a3) then
        if(a1.lt.a2) then
          m=1
        else
          m=2
        end if
      else
        if(a1.lt.a3) then
          m=1
        else
          m=3
        end if
      end if
      k0=0
      1010 prev=ttl
      tgl=next(m,tgl)
      m2=markint(tgl,prev,next)
      m3=mod(m2,3)+1
      m1=mod(m3,3)+1
      a3=volum(r(1,np),r(1,sites(m1,tgl)),r(1,sites(m2,tgl)))
      a1=volum(r(1,np),r(1,sites(m2,tgl)),r(1,sites(m3,tgl)))
      if(a1.lt.a3) then
        m=m1
        a=a1
      else
        m=m3
        a=a3
      end if
      k0=k0+1
      if(k0.gt.ntr) then
        print *, '====> triangle not found <===='
        print *, '====> fatal error <===='
        stop 31
      end if
      if(a.lt.0) goto 1010
      return
      end

*
*
      subroutine outint(nstr1,nst1,r,sites)
*
*
      bybod dliq blizualizacii
      sites -work arr. dimension = maxst*6
      common /points/ nst,ntr
      common /llder/ ldr,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
      madst,rdmax0,ldra,lwin,ldrm,msav,ilg,miner,kva
      6      ,lpik,ipog
      6      ,real r(3,1)

```


intim.f

```

logical frame
integer sites(3,1)
save
r1=0.5
c write (7,*) ntr1,nst1
do 220 j1=1,nst1
  if(r(3,j1).lt.0.0.and.r(3,j1).gt.-9 ) r(3,j1)=0
  if(r(3,j1).gt.zmax ) r(3,j1)= zmax
220 continue
read(2)((sites(i3,i4),i3=1,3,i4=1,ntr1)
nt=0
do 712 i1=1,ntr1
  if(frame(sites(i1,1),i1)) goto 712
  if(frame(sites(2,i1),i1)) goto 712
  if(frame(sites(3,i1),i1)) goto 712
  nt=nt+1
712 continue
write (7,*) nt,nst1
write(8,i6) ,
write(8,i6) ,
write(8,i6) ,
write(8,i6) ,
write(8,i6) ,
format(a,i7,a)
16
15 format(a)
do 10 j1=1,nst1
  if(r(1,j1).ge.0.0 .and.r(1,j1).le.1.0) then
    if(r(2,j1).ge.0.0 .and.r(2,j1).le.1.0) then
      write(7,*)r(1,j1),r(2,j1),r(3,j1)
    else
      write(7,*)r1,r1,r(3,j1)
    end if
  else
    write(7,*)r1,r1,r(3,j1)
  end if
10 continue
na=0
do 710 i1=1,ntr1
  if(frame(sites(i1,1),i1)) goto 710
  if(frame(sites(2,i1),i1)) goto 710
  if(frame(sites(3,i1),i1)) goto 710
  write (7,*) sites(i1,i1),sites(2,i1),sites(3,i1)
710 continue
return
end

function funcal(ns,ms,r,tang)
parameter(nfunc=5)
real tang(nfunc,1),phi(nfunc),r(3,1)
call funcal(r(1,ns)-r(1,ns),r(2,ms)-r(2,ms),phi)
funcal=r(3,ns)
do 1 k=1,nfunc
  funcal=funcal+dprod(tang(k,ns),phi(k))
  return
end

subroutine calmrmat,rhs,r2)
parameter(nfunc=5)
real phi(nfunc),rmat(nfunc,nfunc),rhs(nfunc),r2(3)
sum=1.00/(dprod(r2(1),r2(1))+
  * dprod(r2(2),r2(2))+dprod(r2(3),r2(3)))
sum=sum*2
call funcal(r2(1),r2(2),phi)
do 1 k=1,nfunc
  rhs(k)=rhs(k)+(sum*dprod(r2(3),phi(k)))
end

common subprograms for interpolation
subroutine preint(t,next,sites,radcir,tang,work,minst)
real r(1),next(1),sites(1),radcir(1),tang(1),work(1)
call calrad(r,sites,radcir)
call calnor(r,next,sites,radcir,tang,work,minst)
return
end

subroutine calnor(r,next,sites,radcir,tang,mask,minst)
*
* papametry okpuvhocti, oplcahhoj bokpug modeli
* rband - radiuc, nbndp - kol-bo to-ek ha okpuvhocti
* parameter(nbndp=11,rbnd=10.)
* parameter(nfunc=5)
* integer sh,st
* common /points/ nst,ntr
* real r(3,1)
* integer next(3,1),sites(3,1)
* radcir - kbadpat paduca oplcahhoj bokpug tp-ka okpuvhocti
* real radcir(1)
* real tang(nfunc,1)
* common /llder/ lfo,isu,nml0,xmax,xmin,ymax,ymin,zmax,zmin,grid,
* madst,rdmax0,ldra,lwin,ldrm,msav,ilg,minor,kva
*
* ipik,ipog
* integer mask(1),crv
* mask - work arr. dim = minst
* real r2(3),rnor(nfunc,nfunc)
* logical frame
* sh(m)=mod(m,3)+1
* save

```

59

intim.f

6

```

1010      list=nt
          crv=0
          if (crv.gt.minst/8) then
            print *, 'fatal error'
            stop 21
          end if
          crv=crv+1
          m3=markint(neb,list,next)
          m1=sh(m3)
          m2=sh(m1)
          list=neb
          neb=next(m2,neb)
          nmem(crv)=sites(m3,list)
          rmem(crv)=(r(1,st)-r(1,nmem(crv)))**2
          rmem(crv)=(r(2,st)-r(2,nmem(crv)))**2
          if (list.ne.nt) goto 1010
          b1=0
          b2=0
          do 20 i=1,crv
            r2=(r(1,st)-r(1,nmem(i)))**2
            r2=(r(2,st)-r(2,nmem(i)))**2
            if (r2.lt.3*rdmax0**2) then
              rcorr=rdmax0**2*0.1
              if (r2.gt.rcorr/16) then
                rd=2*sqrt(r2/rcorr)
                rd=sqrt(r2/rcorr)-0.25
              else
                rd=0
              end if
              if (lsmr.eq.1) then
                den=1/(w(st)*(w(st)+w(nmem(i)))*rd)
              else
                den=1/(w(1)*(w(1)+w(i))*rd)
              end if
              b1=b1+rd*(nmem(i))*den
              b2=b2+den
            end if
          end do
          if (lsmr.eq.1) then
            temp(st)=(r(3,st)+w(st)*b1)/(1+w(st)*b2)
          else
            temp(st)=(r(3,st)+w(1)*b1)/(1+w(1)*b2)
          end if
          else
            temp(st)=r(3,st)
          end if
        end if
      continue
    do 50 no=1,nst-nbndp
      if (abs(temp(n0)-r(3,no)).le.3*w(n0)) then
        r(3,no)=temp(n0)
      end if
    end do
    return
  end

```

integer function markint(tr1,tr2,next)

integer tr1,tr2

```

integer next(3,1)
opdelehtie opdehtacil b tp-ke tr1 po tp-ku tr2
do 10 markint=1,3
  if (next(markint,tr1).eq.tr2) return
10 continue
  print *, 'next is in disaccord'
  print *, 'fatal error'
  stop 45
end

subroutine calrad(r,sites,radclr)
  by-icq'c'q'p'adilycy opicahnyx bokpyg tpeygoixhikob okpyvhoctej
  integer tgl,st1,st2,st3
  common /points/ nst,ntr
  real r(3,1)
  integer sites(3,1)
  real radclr(1)
  real a(2,2),b(2,2),det,pp
  real center(2)
  save
  pp(x)=dprod(x,x)
  do 10 tgl=1,ntr
    st1=sites(1,tgl)
    st2=sites(2,tgl)
    st3=sites(3,tgl)
    a(1,1)=r(1,st2)-r(1,st1)
    a(1,2)=r(2,st2)-r(2,st1)
    a(2,1)=r(1,st3)-r(1,st1)
    a(2,2)=r(2,st3)-r(2,st1)
    b(1,1)=pp(r(1,st2))*pp(r(2,st2))-pp(r(1,st1))*pp(r(2,st1))/2
    b(2,1)=pp(r(1,st3))*pp(r(2,st3))-pp(r(1,st1))*pp(r(2,st1))/2
    det=a(1,1)*a(2,2)-a(1,2)*a(2,1)
    center(1)=(a(2,2)*b(1,1)-a(1,2)*b(2,1))/det
    center(2)=(a(1,1)*b(2,1)-a(2,1)*b(1,1))/det
    radclr(tgl)=pp(r(1,st1)-center(1))+pp(r(2,st1)-center(2))
  10 continue
  return
end

subroutine smstat(lstat,r,wght,next,sites,minst,w)
  integer w(3,minst)
  real r(3,1),wght(1)
  integer next(3,1),sites(3,1)
  lstat=1
  id=itest+minst/8
  lsmot=id+minst/8
  ic=lsmot+minst
  line1=ic+minst/4+1
  imask=line1+minst/8
  call smstat(lstat,r,wght,next,sites,w(lstat),w(id),w(lsmot),
    w(ic),w(line1),w(imask),minst)
  return
end

subroutine smstat(lstat,r,w,next,sites,tek,d,smot,c,nel,mask,
  minst)

```

```

1000 1st=nt
      crv=0
      if (crv.gt.minst/8) then
        print *, 'fatal error <---'
        stop 21
      end if
      crv=crv+1
      m3=markint(neb,1st,next)
      m1=sh(m3)
      m2=sh(m1)
      1st=neb
      neb=next(m2,neb)
      nmem(crv)=sites(m3,1st)
      rmem(crv)=(r(1,1st)-r(1,nmem(crv)))**2
               +(r(2,1st)-r(2,nmem(crv)))**2
      if (1st.ne.nt) goto 1010
      b1=0
      b2=0
      do 20 ic=1,crv
        r2=(r(1,1st)-r(1,nmem(ic)))**2
             +(r(2,1st)-r(2,nmem(ic)))**2
        if (r2.lt.3*rdmax0**2) then
          rcorr=rdmax0**2*0.1
          if (r2.gt.rcorr/16) then
            rd=2*sqrt(r2/rcorr)
            rd=sqrt(r2/rcorr)-0.25
          else
            rd=0
          end if
          if (1st.eq.1) then
            den=1/(w(1st)+w(1st)*w(nmem(ic)))*rd
          else
            den=1/(w(1st)+w(1st)*w(1))*rd
          end if
          b1=b1+r(3,nmem(ic))*den
          b2=b2+den
        end if
      continue
      if (1st.eq.1) then
        temp(st)=(r(3,1st)+w(1st)*b1)/(1+w(1st)*b2)
      else
        temp(st)=(r(3,1st)+w(1st)*b1)/(1+w(1st)*b2)
      endif
      else
        temp(st)=r(3,1st)
      end if
    end if
  30 continue
  40 continue
  do 50 n0=1,nst-nbndp
    if (abs(temp(n0)-r(3,n0)).le.3*w(n0)) then
      r(3,n0)=temp(n0)
    end if
  50 continue
  return
end

integer function markint(tr1,tr2,next)
integer tr1,tr2

```

```

integer next(3,1)

opdelehole opiehtaci b tp-ke tr1 po tp-ku tr2
do 10 markint=1,3
  if (next(markint,tr1).eq.tr2) return
10 continue
  print *, 'next is in disaccord <---'
  print *, 'fatal error <---'
  stop 45
end

subroutine calrad(r,sites,radcfr)
  by-lclq 'c'q padiyey opicahhyx bokpyg tpeygoxlkhikob okpyvhoctej
  integer tgl,st1,st2,st3
  common /points/ nst,ntr
  real r(3,1)
  integer sites(3,1)
  real radcfr(1)
  real*8 a(2,2),b(2,2),det,pp
  real center(2)
  save
  pp(x)=dprod(x,x)
  do 10 tgl=1,ntr
    st1=sites(1,tgl)
    st2=sites(2,tgl)
    st3=sites(3,tgl)
    a(1,1)=r(1,st2)-r(1,st1)
    a(1,2)=r(2,st2)-r(2,st1)
    a(2,1)=r(1,st3)-r(1,st1)
    a(2,2)=r(2,st3)-r(2,st1)
    b(1,1)=(pp(r(1,st2))+pp(r(2,st2))-pp(r(1,st1))-pp(r(2,st1)))/2
    b(2,1)=(pp(r(1,st3))+pp(r(2,st3))-pp(r(1,st1))-pp(r(2,st1)))/2
    det=a(1,1)*a(2,2)-a(1,2)*a(2,1)
    center(1)=(a(2,2)*b(1,1)-a(1,2)*b(2,1))/det
    center(2)=(a(1,1)*b(2,1)-a(2,1)*b(1,1))/det
    radcfr(tgl)=pp(r(1,st1)-center(1))+pp(r(2,st1)-center(2))
  10 continue
  return
end

subroutine smstat(istat,r,wight,next,sites,minst,w)
  integer w(3*minst)
  real r(3,1),wight(1)
  integer next(3,1),sites(3,1)
  ltek=1
  id=ltek*minst/8
  lsmot=id+minst/8
  lc=lsmot+minst
  inel=lc+minst/4+1
  imask=inel+minst/8
  call smstat(istat,r,wight,next,sites,w(ltek),w(id),w(lsmot),
               w(lc),w(inel),w(imask),minst)
  return
end

subroutine smstat(istat,r,w,next,sites,tek,d,smot,c,nel,mask,
                  minst)

```

```

smstal cha4ala fopmipuet maccib tek, pspbye ntek ilemehtob
k-pogo codepbat zha-ehiq z-koopdihat b camoj to-ke i ee co-
cedqx i popqda.
parameter (nbnd=11,rbnd=10.)
integer sh,crv,st
real r(3,1),w(1)
integer next(3,1),sites(3,1)
common /points/nst,ntr
common /llder/ lfo,lsu,nml0,xmax,xmin,ymax,ymin,zmax,zmin,grid,
madst,rdmax0,ldra,lwin,ldrm,msav,llg,miner,kva
,lpik,ipog
dimension tek(minst/8),d(mlnst/8),
c(minst/4+1),nel(minst/8)
integer mask(1)
real smot(1)
sh(m)=mod(m,3)+1
save
do 10 ns=1,nst
if (ns.lt.nst-nbndp) then
mask(ns)=0
else
mask(ns)=1
end if
continue
do 80 nt=1,ntr
do 90 i1=1,2
do 80i j1=1,3
r=r+(1,sites(j),nt))
if (r.lt.10.) or .rr.gt. (1.) go to 80
continue
do 70 i0=1,3
st=sites(i0,nt)
if (mask(st).eq.0) then
obo%: i bokpug bep:by sites(i0,nt)
neb=next(sh(i0),nt)
lst=nt
crv=0
crv=crv+1
if (neb.eq.0) goto 80
if (crv.gt.minst/8) then
print *, ' ===> fatal error <=== '
stop 21
end if
m3=markint(neb,lst,next)
n1=sh(n3)
r2=sh(ml)
zapomaha'icq homepa cocodej
nel( crv)=sites(n3,neb)
he o-ehx p'oxoe mecto
if (next(ml,neb).lt.1) goto 80
nel( crv)=sites(markint(next(ml,neb),neb),next(ml,neb))
lst=neb
neb=next(m7,neb)
if (lst.ne.nt) goto 1010
1010
mask(st)=1
ntex=0
do i17 inst=1,crv
cocodej coc-itali. poi11 c-itatx maccib zha-ehi b coc. to-kax

```

```

* itak : center - mediana bybopki a(n)
do 1 i=1,n
  c(i)=(a(i)-center)**2
* c(n) - habop xbadpob otklonehiy elementob a(i) ot mediany
call sort(c,n)
sko=0.
if(n.ge.4) then
  ecil ob'em bybopki n >= 4, to otpacybaem n/4 kpa'jhi xha-
  -eni; cbepxu i ctoixko ve chizu:
    n4=n/4
    do 112 i=n4+1,n-n4-1
      sko=sko+c(i)
      sko=sko/(n-n4)
    else
      if(n.eq.3) then
        ecil ob'em bybopki n = 3, to otpacybaem odho kpa'jhee zha-
        -enie cbepxu i odho chizu:
          sko=c(2)
        else
          ecil ob'em bybopki n = 2, to bepem polucummu:
            sko=0.5*(c(1)+c(2))
          end if
        end if
      else
        call sort(d,i)
        sko=0.
        if(i.ge.4) then
          ecil ob'em bybopki i >= 4, to otpacybaem 1/4 kpa'jhi xha-
          -eni; cbepxu i ctoixko ve chizu:
            i4=i/4
            do 212 i=i4+1,i-i4-1
              sko=sko+d(i)
              sko=sko/(i-i4-14)
            else
              if(i.eq.3) then
                ecil ob'em bybopki i = 3, to otpacybaem odho kpa'jhee zha-
                -enie cbepxu i odho chizu:
                  sko=d(2)
                else
                  ecil ob'em bybopki i = 2, to bepem polucummu:
                    sko=0.5*(d(1)+d(2))
                  end if
                end if
              else
                call sort(sko)
              end if
            end if
          end if
        end if
      end if
    end if
  end if
* po agnem papameti p pabnym tpey by-iclenhym ep.-xbadp. otklonehiqm:
* ia.en pooditcay minimizaciy funkciy xx'bepa:
  n=sko/3.
do 2 i=1,n
  m=1.
  c(i)=a(i)-n
  -eni;
  center= argmin { f(x), x=1,...,n }
  minimym bepetcay po skal. belii-ibe t.
  fo - funkciy xx'bepa, by-iclenetcay b function ro.
end

```

```

c(m)=a(i)+h
continue
call sort(c,m)
i1=1
i2=m
del=de(c(i),a,n,h)
de2=de(c(m),a,n,h)
if(i2-i1.le.1) go to 9
is=(i1+i2)/2
des=de(c(is),a,n,h)
if(des < de2) then
  center=c(is)
return
4 i1=is
del=des
go to 8
5 i2=is
de2=des
go to 8
9 continue
center=c(i1)-(c(i2)-c(i1))*del/(de2-del)
* itak, ickomaq bybopcocto'j-iba q ochenka cpednego zha-ehiq
* bybopki a(1)...a(n) by-iclenha i pabha center
return
end
real function de(t,a,n,h)
real a(n),b,c,ta
de=0.
do 1 i=1,n
  ta=t-a(i)
  if(abs(ta).gt.h) then
    if(ta.gt.h) then
      de=de+h
    else
      de=de-h
    end if
  else
    de=de+ta
  end if
1 continue
return
end
subroutine sort(a,n)
* podpogpamna sort delaet coptipobnu maccika a(n) po bo'pactah'
real a(n)
if(n.eq.1) return
do 1 i=1,n-1
  do 1 j=i+1,n
    if(a(i)-a(j)) 1,1,1
    t=a(i)
    a(i)=a(j)
    a(j)=t
  1 continue
  return
end
subroutine GAUSSJ(A,N,NP,B,M,MP)
  implicit real*8(a-h,o-z)
  parameter (NMAX=50)
  real A(NP,NP),B(NP,MP)
  integer IPIV(NMAX),INDXR(NMAX),INDXC(NMAX)
  do 11 j=1,N

```

intim.f

9

```

11 IPIV(J)=0
   CONTINUE
DO 22 I=1,N
   BIG=0.
DO 13 J=1,N
   IF(IPIV(J).NE.1)THEN
DO 12 K=1,N
   IF(IPIV(K).EQ.0) THEN
   IF (ABS(P(J,K)).GE.BIG)THEN
     BIG=ABS(A(I,J,K))
     IROW=J
     ICOL=K
   ENDIF
ELSE IF (IPIV(K).GT.1) THEN
   DO 48 KK=1,N
     B(KK,I)=0
   RETURN
   CONTINUE
ENDIF
CONTINUE
12 CONTINUE
13 CONTINUE
IPIV(ICOL)=IPIV(ICOL)+1
IF (IROW.NE.ICOL) THEN
DO 14 L=1,N
   DUM=A(IROW,L)
   A(IROW,L)=A(ICOL,L)
   A(ICOL,L)=DUM
CONTINUE
DO 15 L=1,M
   DUM=B(IROW,L)
   B(IROW,L)=B(ICOL,L)
   B(ICOL,L)=DUM
CONTINUE
ENDIF
14 CONTINUE
15 CONTINUE
INDXR(I)=IROW
INDXC(I)=ICOL
IF (A(ICOL,ICOL).EQ.0.) THEN
   DO 88 KK=1,N
     B(KK,I)=0
   RETURN
ENDIF
PIVINV=1./A(ICOL,ICOL)
A(ICOL,ICOL)=1.
DO 16 L=1,N
   A(ICOL,L)=A(ICOL,L)*PIVINV
CONTINUE
DO 17 L=1,M
   B(ICOL,L)=B(ICOL,L)*PIVINV
CONTINUE
DO 21 LL=1,N
   IF(LL.NE.ICOL)THEN
     DUM=A(LL,ICOL)
     A(LL,ICOL)=0.
DO 18 L=1,N
     A(LL,L)=A(LL,L)-A(ICOL,L)*DUM
CONTINUE
DO 19 L=1,M
     B(LL,L)=B(LL,L)-B(ICOL,L)*DUM
CONTINUE
ENDIF
21 CONTINUE
22 CONTINUE
DO 24 L=N,1,-1
   IF(INDXR(L).NE.INDXC(L))THEN

```

ints.f

1

```

subroutine ints(ind5, ind6)
*
*   interpolate by-iclenie koopdihat hobyx to-ek
*   parameter(minst=70000, maxst=minst, maxtr=2*maxst)
*   integer next, sites, tgl
*   'nst' - # to-ek, 'r' - ix koopdihat
*   common /points/ nst, r(4, maxst)
*   'ntr' - # tpeugolxhikob, '/net/' - cpicok cmevhocti dlog
*   qafa dualxhogo k tplahulqeli
*   common /net/ ntr, next(3, maxtr), sites(3, maxtr)

if(ind5 .eq. 1) then
  open(8, file='text')
  write(8,*) '50 50'
  write(8,*) 'Interpolation is not aloud'
  write(8,*) 'Second input parameter is too small'
  close(8)
  call try1('text')
  return
end if

open(1, file='stempor', form='unformatted', status='unknown')
open(2, file='saddul', form='unformatted', status='unknown')
open(7, file='triang')

read (2, end=1010, err=1010) ntr, nstl
read (2, end=1010, err=1010) ((r(i0,1), i0=1, 4), i1=1, nstl)
print *, '====>', nstl, ' points have been read <===='
if(nstl.gt.maxst) then
  print *, '====> maxst too small, change and recompile <===='
  stop 11
end if
goto 1020

1010 print *, '====> error reading <===='
stop 12
1020 continue

read (1, end=1010, err=1010) ntr, nst0
read (1, end=1030, err=1030) ((sites(i4,13), i4=1, 3), i5=1, ntr)
read (1, end=1030, err=1030) ((next(i2,13), i2=1, 3), i3=1, ntr)
print *, '====>', ntr, ' triangles have been read <===='
if(ntr.gt.maxtr) then
  print *, '====> maxtr too small, change and recompile <===='
  stop 13
end if
goto 1040

1030 print *, '====> error reading <===='
stop 14
1040 continue
if(nst0.gt.minst) then
  print *, '====> minst too small, change and recompile <===='
  stop 15
end if
if(nst0.ge.nstl) then
  print *, '====> there are no added points <===='
  stop 16
end if

nst=nst0
by-iclenie papametpob iteppolqclohnyx funkci)
call coef
cop.ipobka to-ek po tpeugolxhikam
print *, '====> sorting <===='

```

```

call prepar(ntr, nstl)
print *, '====> interpolation <===='
by-iclenie koopdihat hobyx to-ek
nstl - hoboe polhoe k-bo to-ek pobepxhocti
nst0 - ctapoe k-bo to-ek pobepxhocti
call addpts(ntrl, nstl)

ccc
print *, '====> ok' i write <===='
if(ind6.eq.0) then
  call outint(ntrl, nstl)
end if
if(ind6.eq.1) then
  call outint1(ntrl, nstl)
end if
close(1)
close(2)
return
end

subroutine coef
*
*   by-iclenie pepepob b bep(lhax opophoj pe(ekki
*   parameter(minst=70000, maxst=minst, maxtr=2*maxst)
*   integer next, sites, sh, crv, st
*   common /points/ nst, r(4, maxst)
*   common /net/ ntr, next(3, maxtr), sites(3, maxtr)
*   real rnor(3), tang(3), sig(3), r1(3), r2(3)
*   common /reps/ e(3,3, minst), der(2,3, minst)
*   common mask(minst)
*   sh(m)=mod(m,3)+1
*   save

do 10 ns=1, nst
  mask(ns)=0
  continue
do 100 nt=1, ntr
  do 90 lo=1, 3
    st=sites(i0, nt)
    if(mask(st) .eq. 0) then
      obojti bokpug bep(lhy sites(i0, nt)
      rnorm - hopmalx k kacatelxhoj plochocti
      do 20 ko=1, 3
        rnor(ko)=0
        continue
      neb=next(sh(i0), nt)
      lst=nt
      crv=0
      crv=crv+1
      if(crv.gt.minst/4) then
        print *, '====> fatalxhaq o(ibka <===='
        stop 21
      end if
      m3=mark(neb, lst)
      m1=sh(m3)
      m2=sh(m1)
      by-iclenieq hopmalx k tpeugolxhiku
      if(crv.eq.1) then
        r1(1)=r(1, sites(m2, neb))-r(1, st)
        r1(2)=r(2, sites(m2, neb))-r(2, st)
        r1(3)=r(3, sites(m2, neb))-r(3, st)
        absr1=sqrt((r1(1)**2+r1(2)**2+r1(3)**2)
        r1(1)=r1(1)/absr1
        r1(2)=r1(2)/absr1
        r1(3)=r1(3)/absr1

```



```

else
  r1(1)=r2(1)
  r1(2)=r2(2)
  r1(3)=r2(3)
  end if
  r2(1)=r(1,sites(m3,neb))-r(1,st)
  r2(2)=r(2,sites(m3,neb))-r(2,st)
  r2(3)=r(3,sites(m3,neb))-r(3,st)
  absr2=sqrt(r2(1)**2+r2(2)**2+r2(3)**2)
  r2(1)=r2(1)/absr2
  r2(2)=r2(2)/absr2
  r2(3)=r2(3)/absr2
  rnor(1)=rnor(1)+
    (r1(2)*r2(3)-r1(3)*r2(2)-rnor(1))/crv
  rnor(2)=rnor(2)+
    (r1(3)*r2(1)-r1(1)*r2(3)-rnor(2))/crv
  rnor(3)=rnor(3)+
    (r1(1)*r2(2)-r1(2)*r2(1)-rnor(3))/crv
  lst=neb
  neb=next(m2,neb)
  if (lst.ne.nt) goto 1010
  e = pepep b bep(1)e
  jmin=0
  cmin=1
  do 30 j1=1,3
    e(1,j1,st)=r(j1,st)
    if(abs(e(1,j1,st)).lt.cmin) then
      jmin=j1
      cmin=abs(e(1,j1,st))
    end if
    sig(j1)=0
    continue
  30 sig(jmin)=1
  do 40 j2=1,3
    tang(j2)=e(1,sh(j2),st)*sig(sh(j2+1))
    -e(1,sh(j2+1),st)*sig(sh(j2))
    continue
  40 gl=sqr(scal(tang,tang))
  do 50 j3=1,3
    e(3,j3,st)=tang(j3)/gl
    continue
  50 e(2,j4,st)=e(3,sh(j4),st)*e(1,sh(j4+1),st)
    -e(3,sh(j4+1),st)*e(1,sh(j4),st)
    continue
  60 do 80 j6=1,3
    tang(j6)=0
    do 70 j5=1,3
      tang(j6)=tang(j6)+e(j6,j5,st)*rnor(j5)
    continue
  70 continue
  80 if(tang(1).gt.1e-10) then
    deriv(2,st)=tang(2)/tang(1)*r(4,st)
    deriv(3,st)=tang(3)/tang(1)*r(4,st)
  else
    deriv(2,st)=tang(2)*le10
    deriv(3,st)=tang(3)*le10
    print *, '====> very large derivatives <===='
    end if
  end if
  continue
100 continue
return
end

```

```

subroutine prepar(ntr,nst)
  ha-alxhaq podgotobka
  parameter(minst=70000,maxst=minst,maxtr=2*maxst)
  'nst' - # to-ex, 'r' - ix koopdihaty, 'w' - pogpe(hocti.
  common /points/ nst0,r(4,maxst)
  common /bink/ lord(maxst),list(0:maxtr),jord(maxst)

  ntq=1
  nd=nst-nst0
  do 10 np=1,nd
    call searin(ntq,np+nst0)
    jord(np)=ntq
    10 continue
    call kb07ai(jord,nd,lord)
    nt=ntr
    list(nt)=nst
    do 20 j3=nst,nst0+1,-1
      ns=j3-nst0
      lord(j3)=lord(ns)+nst0
      1010 if(jord(ns).eq.nt) goto 1020
      nt=nt-1
      if(nt.lt.0) then
        print *, '% abend in ''prepar''
        stop 21
      end if
      list(nt)=j3
      goto 1010
    20 continue
    do 30 k0=0,nt-1
      list(k0)=nst0
    30 continue
    do 40 j4=1,nst0
      lord(j4)=j4
    40 lord(j4)=j4
    return
  end

subroutine adpts(ntr1,nst1)
  parameter(minst=70000,maxst=minst,maxtr=2*maxst)
  integer next,sites,st,sh
  common /points/ nst,r(4,maxst)
  common /net/ ntr,next(3,maxtr),sites(3,maxtr)
  common /reps/ e(3,3,minst)
  common /bink/lord(maxst),list(0:maxtr),jord(maxst)
  real h2(3),q(3)
  real a(4,3),c(6,3)
  sh(m)=mod(m,3)+1
  save

  ns=list(0)
  do 50 nt=1,ntr
    nds=list(nt)-ns
    if(nds.gt.0) then
      call affn(nt,a,c)
      nset=nt
      do 40 k0=1,nds
        np=lord(k0+ns)
        call searin(nset,np)
        if(nset.ne.nt) then
          print *, 'bep(1)ha',ns+k0, ' he e tpeugolxhiku',nset

```


ints.f

4

```

common /net/ ntr,next(3,maxtr),sites(3,maxtr)
real h(3),fp(3),tgn(3)
sh(m)=mod(m,3)+1
save

st1=sites(1,nt)
st2=sites(2,nt)
st3=sites(3,nt)
do 10 i=1,3
  tgn(i)=
  . (rsh(10),st1)-r(sh(10),st3))*r(sh(10+1),st2)-r(sh(10+1),st3))
  . -(rsh(10+1),st1)-r(sh(10+1),st3))*r(sh(10),st2)-r(sh(10),st3))
10 continue
r1=0
r2=0
r3=0
do 20 m1=1,3
  r1=r1+(r(m1,np)-r(m1,st1))*2
  r2=r2+(r(m1,np)-r(m1,st2))*2
  r3=r3+(r(m1,np)-r(m1,st3))*2
20 continue
if(r1.lt.eps) then
  h(1)=1
  h(2)=0
  h(3)=0
else if(r2.lt.eps) then
  h(1)=0
  h(2)=1
  h(3)=0
else if(r3.lt.eps) then
  h(1)=0
  h(2)=0
  h(3)=1
else
  sc1=0
  sc2=0
  do 30 i1=1,3
    sc1=sc1+r(i1,np)*tgn(i1)
    sc2=sc2+r(i1,st1)*tgn(i1)
  30 continue
  do 40 i2=1,3
    fp(i2)=r(i2,np)*sc2/sc1
    continue
  h1=area: (fp,r(1,st2),r(1,st3))
  h2=area: (fp,r(1,st3),r(1,st1))
  h3=area: (fp,r(1,st1),r(1,st2))
  h(1)=(h2-h3)*2+h3*r3**2+h1
  h(2)=(h3-h1)*2+h1*r1**2+h2
  h(3)=(h1-h2)*2+h2*r2**2+h3
  z=h(1)+h(2)+h(3)
  h(1)=h(1)/z
  h(2)=h(2)/z
  h(3)=h(3)/z
end if
return
if(h(1).lt.h(2)) then
  hm=h2
  im=2
else
  hm=h1
  im=1
end if
end if

if(h(3).lt.h(im)) then
  h(3)=0
  h(im)=1
else
  h(3)=1
  h(im)=0
end if
return
end

subroutine searln(tgl,np)
*
* polck tpeugolxhika b kotopyj popadaet to-ka np
* parameter(minst=70000,maxst=minst, maxtr=2*maxst)
integer next,sites,tgl,m,np,prev
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
save

mk=0
a3=volum(r(1,np),r(1,sites(1,tgl)),r(1,sites(2,tgl)))
a1=volum(r(1,np),r(1,sites(2,tgl)),r(1,sites(3,tgl)))
a2=volum(r(1,np),r(1,sites(3,tgl)),r(1,sites(1,tgl)))
if(a1.ge.0.and.a2.ge.0.and.a3.ge.0) then
  return
end if
if(a2.lt.a3) then
  if(a1.lt.a2) then
    m=1
  else
    m=2
  end if
else
  if(a1.lt.a3) then
    m=1
  else
    m=3
  end if
end if
k0=0
prev=tgl
tgl=next(m,tgl)
m2=mark(tgl,prev)
m3=mod(m2,3)+1
m1=mod(m3,3)+1
a3=volum(r(1,np),r(1,sites(m1,tgl)),r(1,sites(m2,tgl)))
a1=volum(r(1,np),r(1,sites(m2,tgl)),r(1,sites(m3,tgl)))
if(a1.lt.a3) then
  m=m1
  a=a1
else
  m=m3
  a=a3
end if
k0=k0+1
if(k0.gt.maxtr) then
  print *, '==== tpeugolxhik he hajden <===='
  stop 31
end if
if(a.lt.0) goto 1010
return
end

```

ints.f

3

```

end if
call wghts(h2,nt,np)
!htepqolqicq: bzbelehnoe cpadhee ot tpex bepilh
h2(1)+h2(2)+h2(3)=1
rad=0
do 30 nv=1,3
  st=sites(nv,nt)
  do 20 ml=2,3
    q(ml)=0
    do 10 m2=1,3
      q(ml)=q(ml)+e(m1,m2,st)*r(m2,np)
    continue
  t0=asin(sqrt(q(2)**2+q(3)**2))/sqrt(q(2)**2+q(3)**2)
  q(2)=q(2)*t0
  q(3)=q(3)*t0
  x=a(1,nv)*q(2)+a(3,nv)*q(3)
  y=a(2,nv)*q(2)+a(4,nv)*q(3)
  if(x.lt.0) then
    print *, 'error: x=', x
  end if
  if(y.lt.0) then
    print *, 'error: y=', y
  end if
  w=r(4,st)+c(1,nv)+c(3,nv)*x+c(2,nv)+c(4,nv)*y
  *x*y*((c(5,nv)+c(6,nv))/2+
  (c(5,nv)-c(6,nv))/2*(x-y)/sqrt(x**2+y**2))
  rad=rad+w*h2(nv)
  continue
r(4,np)=rad
end if
ns=1ist(nt)
return
end

```

subroutine afflo,

4

```

! if h(3).lt.1.1-h(1)*h(1) then
  h(3)=0
  h(1)=1
else
  h(3)=1
  h(1)=0
end if
rad=0
do 30 nv=1,3
  st=sites(nv,nt)
  do 20 ml=2,3
    q(ml)=0
    do 10 m2=1,3
      q(ml)=q(ml)+e(m1,m2,st)*r(m2,np)
    continue
  t0=asin(sqrt(q(2)**2+q(3)**2))/sqrt(q(2)**2+q(3)**2)
  q(2)=q(2)*t0
  q(3)=q(3)*t0
  x=a(1,nv)*q(2)+a(3,nv)*q(3)
  y=a(2,nv)*q(2)+a(4,nv)*q(3)
  if(x.lt.0) then
    print *, 'error: x=', x
  end if
  if(y.lt.0) then
    print *, 'error: y=', y
  end if
  w=r(4,st)+c(1,nv)+c(3,nv)*x+c(2,nv)+c(4,nv)*y
  *x*y*((c(5,nv)+c(6,nv))/2+
  (c(5,nv)-c(6,nv))/2*(x-y)/sqrt(x**2+y**2))
  rad=rad+w*h2(nv)
  continue
r(4,np)=rad
end if
ns=1ist(nt)
return
end

```

```

! if h(3).lt.1.1-h(1)*h(1) then
  h(3)=0
  h(1)=1
else
  h(3)=1
  h(1)=0
end if
rad=0
do 30 nv=1,3
  st=sites(nv,nt)
  do 20 ml=2,3
    q(ml)=0
    do 10 m2=1,3
      q(ml)=q(ml)+e(m1,m2,st)*r(m2,np)
    continue
  t0=asin(sqrt(q(2)**2+q(3)**2))/sqrt(q(2)**2+q(3)**2)
  q(2)=q(2)*t0
  q(3)=q(3)*t0
  x=a(1,nv)*q(2)+a(3,nv)*q(3)
  y=a(2,nv)*q(2)+a(4,nv)*q(3)
  if(x.lt.0) then
    print *, 'error: x=', x
  end if
  if(y.lt.0) then
    print *, 'error: y=', y
  end if
  w=r(4,st)+c(1,nv)+c(3,nv)*x+c(2,nv)+c(4,nv)*y
  *x*y*((c(5,nv)+c(6,nv))/2+
  (c(5,nv)-c(6,nv))/2*(x-y)/sqrt(x**2+y**2))
  rad=rad+w*h2(nv)
  continue
r(4,np)=rad
end if
ns=1ist(nt)
return
end

```

ints.f

4

```

common /net/ ntr, next(3, maxtr), sites(3, maxtr)
real h(3), rp(3), tgn(3)
sh(m) = mod(m, 3) + 1
save

st1 = sites(1, nt)
st2 = sites(2, nt)
st3 = sites(3, nt)
do 10 i = 1, 3
  tgn(i) =
  . (r(sh(i), st1) - r(sh(i), st2)) * (r(sh(i), st2) - r(sh(i), st3))
  . - (r(sh(i), st1) - r(sh(i), st3)) * (r(sh(i), st2) - r(sh(i), st3))
  10 continue
  r1 = 0
  r2 = 0
  r3 = 0
  do 20 m1 = 1, 3
    r1 = r1 + (r(m1, np) - r(m1, st1)) ** 2
    r2 = r2 + (r(m1, np) - r(m1, st2)) ** 2
    r3 = r3 + (r(m1, np) - r(m1, st3)) ** 2
  20 continue
  if (r1.lt.eps) then
    h(1) = 1
    h(2) = 0
    h(3) = 0
  else if (r2.lt.eps) then
    h(1) = 0
    h(2) = 1
    h(3) = 0
  else if (r3.lt.eps) then
    h(1) = 0
    h(2) = 0
    h(3) = 1
  else
    scl = 0
    sc2 = 0
    do 30 i1 = 1, 3
      scl = scl + r(i1, np) * tgn(i1)
      sc2 = sc2 + r(i1, st1) * tgn(i1)
    30 continue
    do 40 i2 = 1, 3
      rp(i2) = r(i2, np) * sc2 / scl
      continue
      h1 = area2(rp, r(1, st2), r(1, st3))
      h2 = area2(rp, r(1, st3), r(1, st1))
      h3 = area2(rp, r(1, st1), r(1, st2))
      h(1) = (h2 + r2 ** 2 + h3 + r3 ** 2) * h1
      h(2) = (h3 + r3 ** 2 + h1 + r1 ** 2) * h2
      h(3) = (h1 + r1 ** 2 + h2 + r2 ** 2) * h3
      z = h(1) + h(2) + h(3)
      h(1) = h(1) / z
      h(2) = h(2) / z
      h(3) = h(3) / z
    end if
  return
  if (h(1).lt.h(2)) then
    h1m = h2
    i1m = 2
    h(1) = 0
  else
    h1m = h1
    i1m = 1
    h(2) = 0
  end if

```

```

if (h(3).lt.h(1m)) then
  h(3) = 0
  h(1m) = 1
else
  h(3) = 1
  h(1m) = 0
end if
return
end

subroutine searln(tgl, np)
  polck tpeugolxhika b kotopyj popadaet to-ka np
  parameter (minst=70000, maxst=minst, maxtr=2*maxst)
  integer next, sites, tgl, m, np, prev
  common /points/ nst, r(4, maxst)
  common /net/ ntr, next(3, maxtr), sites(3, maxtr)
  save

  mx = 0
  a3 = volum(r(1, np), r(1, sites(1, tgl)), r(1, sites(2, tgl)))
  a1 = volum(r(1, np), r(1, sites(2, tgl)), r(1, sites(3, tgl)))
  a2 = volum(r(1, np), r(1, sites(3, tgl)), r(1, sites(1, tgl)))
  if (a1.ge.0.and.a2.ge.0.and.a3.ge.0) then
    return
  end if
  if (a2.lt.a3) then
    if (a1.lt.a2) then
      m = 1
    else
      m = 2
    end if
  else
    if (a1.lt.a3) then
      m = 1
    else
      m = 3
    end if
  end if
  k0 = 0
  1010 prev = tgl
  tgl = next(m, tgl)
  m2 = mark(tgl, prev)
  m3 = mod(m2, 3) + 1
  m1 = mod(m3, 3) + 1
  a3 = volum(r(1, np), r(1, sites(m1, tgl)), r(1, sites(m2, tgl)))
  a1 = volum(r(1, np), r(1, sites(m2, tgl)), r(1, sites(m3, tgl)))
  if (a1.lt.a3) then
    m = m1
    a = a1
  else
    m = m3
    a = a3
  end if
  k0 = k0 + 1
  if (k0.ge.maxtr) then
    print *, '==== tpeugolxhik he hajdeh <===='
    stop 31
  end if
  if (a.lt.0) goto 1010
  return
end

```

ints.f

5

```

real function area2(x,y,z)
*
* plojadx tpeugolxhika b kbadrate
*
area2 = [x,y]**2/4
real x(3),y(3),z(3)
area2=0
do 10 m=1,3
  n=mod(m,3)+1
  area2=area2+((x(m)-z(m))*(y(n)-z(n))-(x(n)-z(n))*(y(m)-z(m)))**2
10 continue
area2=area2/4
return
end

subroutine outint(ntrl,nstl)
*
* bybod diq blizualizacii
*
parameter(minst=70000,maxst=minst, maxtr=2*maxst)
integer sites
common /points/ nst,r(4,maxst)
common /repsers/ mord(maxst)
common /bink/iord(maxst),list(0:maxtr),jord(maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
save

write(7,*) ntrl,nstl
do 10 j2=1,nstl
  write(7,*) ((r(i1,iord(j2))*r(4,iord(j2))),i1=1,3)
10 continue
do 20 i2=1,nstl
  mord(iord(i2))-i2
20 continue
read(2)((sites(i1,j2),i1=1,3),j2=1,ntrl)
do 11 j2=1,ntrl
  write(7,*) (mord(sites(i1,j2)),i1=1,3)
11 continue
close(7)
return
end

```

```

subroutine outint1(ntrl,nstl)
*
* bybod diq blizualizacii
*
parameter(minst=70000,maxst=minst, maxtr=2*maxst)
integer sites
common /points/ nst,r(4,maxst)
common /repsers/ mord(maxst)
common /bink/iord(maxst),list(0:maxtr),jord(maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
save

```

```

write(7,*) ntrl,nstl
do 10 j2=1,nstl
  write(7,*) ((r(i1,iord(j2))*r(4,iord(j2))),i1=1,3)
10 continue
do 20 i2=1,nstl
  mord(iord(i2))-i2
20 continue
read(2)((sites(i1,j2),i1=1,3),j2=1,ntrl)
do 11 j2=1,ntrl
  write(7,*) (mord(sites(i1,j2)),i1=1,3)
11 continue
close(7)
return
end

```

```

subroutine outint1(ntrl,nstl)
*
* bybod diq blizualizacii
*
parameter(minst=70000,maxst=minst, maxtr=2*maxst)
integer sites
common /points/ nst,r(4,maxst)
common /repsers/ mord(maxst)
common /bink/iord(maxst),list(0:maxtr),jord(maxst)
common /net/ ntr,next(3,maxtr),sites(3,maxtr)
save
read(2)((sites(i1,j2),i1=1,3),j2=1,ntrl)
nt = 0
do 712 it = 1,ntrl
  if(r(4,sites(i1,it)).gt. 1.5) goto 712
  if(r(4,sites(2,it)).gt. 1.5) goto 712
  if(r(4,sites(3,it)).gt. 1.5) goto 712
  nt = nt+1
712 continue

write(7,*) nt,nstl

```

—

```

c1      kb07a      21/04/80
cname kb07a(r)
subroutine kb07a(count,n,index)
c
c      kb07a handles real single-length variables
c standard fortran 66 (a verified pfort subroutine)
c the work-space 'mark' of length 50 permits up to 2**(50/2) numbers
c to be sorted. this is more than the ibm virtual memory space
c will hold.
c dimension count(n),'mark'(50),index(n)
c set index array to original order.
do 10 i=1,n
index(i)=i
10 continue
c check that a trivial case has not been entered.
if(n.eq.1)goto 200
if(n.ge.1)go to 30
write(6,20)
20 format(///20x,65h '**kb07a*** no numbers to be sorted ** return to k
2 calling program )
goto 200
c 'm' is the length of segment which is short enough to enter
the final sorting routine. it may be easily changed.
30 m=12
c set up initial values.
is=2
is=1
if=n
do 190 mloop=1,n
c if segment is short enough sort with final sorting routine.
ifka=if-is
if(((ifka+1).gt.m))goto 70
if(((ifka+1).gt.m))goto 70
c ..... final sorting ....
c ( a simple bubble sort )
isl=is+1
do 60 j=isl,if
i=j
40 if(count(i-1).lt.count(i))goto 60
if(count(i-1).gt.count(i))goto 50
if(index(i-1).lt.index(i))goto 60
50 av=count(i-1)
count(i-1)=count(i)
count(i)=av
int=index(i-1)
index(i-1)=index(i)
index(i)=int
i=i-1
if(i.gt.is)goto 40
60 continue
is=is+2
goto 170
c ..... quicksort .....
select the number in the central position in the segment as
the test number. replace it with the number from the segment's
highest address.
70 ly=(is+if)/2
x=count(iy)
intest=index(iy)
count(iy)=count(if)
index(iy)=index(if)
the markers '1' and 'ifka' are used for the beginning and end
of the section not so far tested against the present value
of x.

```

kb07ai.f

1

c subroutine kb07ai(count,n,index)

c kb07a handles real single-length variables
c standard fortran 66 (a verified pfort subroutine)

c the work-space 'mark' of length 50 permits up to 2*(50/2) numbers
c to be sorted. this is more than the ibm virtual memory space

c will hold.
c implicit integer (a-z)
c integer count(n),mark(50),index(n)
c set index array to original order.

do 10 i=1,n
index(i)=i
10 continue

c check that a trivial case has not been entered.

if(n.eq.1)goto 200
if(n.ge.1)go to 30
write(6,20)

20 format('///20x,65h ***kb07a*** no numbers to be sorted ** return to
2 calling program')

goto 200

c 'm' is the length of segment which is short enough to enter
c the final sorting routine. it may be easily changed.

30 m=12

c set up initial values.

la=2

is=1

if=n

do 190 mloop=1,n

c if segment is short enough sort with final sorting routine.

ifka=if-is

if((ifka+1).gt.m)goto 70

c***** final sorting *****

c (a simple bubble sort)

lsl=ls+1

do 60 j=isl,if

l=j

40 if(count(l-1).lt.count(l))goto 60

if(count(l-1).gt.count(l))goto 50

if(index(l-1).lt.index(l))goto 60

50 av=count(l-1)

count(l-1)=count(l)

count(l)=av

int=index(l-1)

index(l-1)=index(l)

index(l)=int

l=j-1

if(l.gt.is)go to 40

60 continue

la=la-2

goto 170

c***** quicksort *****

select the number in the central position in the segment as
the test number. replace it with the number from the segment's
highest address.

70 ly=(ls+if)/2

x=count(ly)

intest=index(ly)

count(ly)=count(if)

index(ly)=index(if)

the markers 'l' and 'ifk' are used for the beginning and end
of the section not so far tested against the present value
of x.

k=1

ifk=if

c we alternate between the outer loop that increases l and the
c inner loop that reduces ifk, moving numbers and indices as
c necessary, until they meet.

do 110 l=1,if

if(x.gt.count(l))goto 110

if(x.lt.count(l))goto 80

if(intest.gt.index(l))goto 110

80 if(l.ge.ifk)goto 120

count(ifk)=count(l)

index(ifk)=index(l)

kl=k

do 100 k=kl,ifka

ifk=if-k

if(count(ifk).gt.x)goto 100

if(count(ifk).lt.x)goto 90

if(intest.le.index(ifk))goto 100

90 if(l.ge.ifk)goto 130

count(l)=count(ifk)

index(l)=index(ifk)

go to 110

100 continue

goto 120

110 continue

c return the test number to the position marked by the marker
c which did not move last. it divides the initial segment into
c 2 parts. any element in the first part is less than or equal
c to any element in the second part, and they may now be sorted
c independently.

120 count(ifk)=x

index(ifk)=intest

ip=ifk

goto 140

130 count(l)=x

index(l)=intest

ip=1

c store the longer subdivision in workspace.

140 if((ip-is).gt.(if-ip))goto 150

mark(la)=if

mark(la-1)=ip+1

if=ip-1

goto 160

150 mark(la)=ip-1

mark(la-1)=is

is=ip+1

c find the length of the shorter subdivision.

160 lngth=if-is

if(lngth.le.0)goto 180

c if it contains more than one element supply it with workspace.

la=la+2

goto 190

170 if(la.le.0)goto 200

c obtain the address of the shortest segment awaiting quicksort

180 if=mark(la)

is=mark(la-1)

190 continue

200 return

end

kb000660
kb000670
kb000680
kb000690
kb000700
kb000710
kb000720
kb000730
kb000740
kb000750
kb000760
kb000770
kb000780
kb000790
kb000800
kb000810
kb000820
kb000830
kb000840
kb000850
kb000860
kb000870
kb000880
kb000890
kb000900
kb000910
kb000920
kb000930
kb000940
kb000950
kb000960
kb000970
kb000980
kb000990
kb001000
kb001010
kb001020
kb001030
kb001040
kb001050
kb001060
kb001070
kb001080
kb001090
kb001100
kb001110
kb001120
kb001130
kb001140
kb001150
kb001160
kb001170
kb001180
kb001190
kb001200
kb001210
kb001220

lib.f

1

```

subroutine pllder(name)
character*8 name
common /llder/ ifo, isu, nm10, xmax, xmin, ymax, ymin, zmax, zmin, grid,
, madst, rdmax0, idra, lwin, idrm, msav, ilg, miner, kva
, ipix, ipog
save
open(4, file='leader')
write(4,*) '.....'
write(4,*) '..... Information about the execution .....
write(4,2) name
format(1x, '..... the program ', a8, ' has run .....
write(4,*) '.....'
write(4,*) '..... Is the input file formatted or unformatted ?(f/u)'
if(lfo.ne.0) write(4,1) 'f'
if(lfo.eq.0) write(4,1) 'u'
write(4,*) '.....'
, ' does the input file contain errors ?(y/n)'
if(lpog.ne.0) write(4,1) 'y'
if(lpog.eq.0) write(4,1) 'n'
write(4,*) '.....'
, ' Is this surface or function with several values ?(s/f)'
if(lisu.eq.0) write(4,1) 'f'
if(lisu.ne.0) write(4,1) 's'
if(lisu.ne.0) goto 20
write(4,*) ' how many values has the function ? '
write(4,*) nm10
write(4,*) ' minimal and maximal values of x '
write(4,*) xmin, xmax
write(4,*) ' minimal and maximal values of y '
write(4,*) ymin, ymax
write(4,*) ' minimal and maximal values of z '
write(4,*) zmin, zmax
write(4,*) ' the size of grid for averaging'
write(4,*) grid
if(lisu.eq.0) goto 30
write(4,*) ' the maximal number of points in the model'
write(4,*) madst
write(4,*) ' the radius of confidence'
write(4,*) rdmax0
write(4,*) ' what smoothing do you prefer ? (n/1/2/q)'
write(4,*) ' n - no smoothing '
write(4,*) ' 1 - smooth according to errors (from the file)'
write(4,*) ' 2 - smooth (all errors = 1)'
write(4,*) ' q - request during the execution'
if(kva.eq.0) write(4,1) 'n'
if(kva.eq.1) write(4,1) '1'
if(kva.eq.2) write(4,1) '2'
if(kva.lt.0) write(4,1) 'q'
write(4,*) ' kill the peaks ? (n/1/2/q):'
write(4,*) ' n - no'
write(4,*) ' 1 - kill according to errors (from the file)'
write(4,*) ' 2 - kill by statistical method'
write(4,*) ' q - request during the execution'
if(ipik.eq.0) write(4,1) 'n'
if(ipik.eq.1) write(4,1) '1'
if(ipik.eq.2) write(4,1) '2'
if(ipik.lt.0) write(4,1) 'q'
write(4,*) ' draw the triangulation during the execution? (y/n/q)'
if(idra.eq.0) write(4,1) 'n'
if(idra.gt.0) write(4,1) 'y'
if(idra.lt.0) write(4,1) 'q'
if(lisu.ne.0) goto 77
write(4,*) ' will you work with windows ?'
if(lwin.eq.0) write(4,1) 'n'
if(lwin.ne.0) write(4,1) 'y'
write(4,*) ' draw the maps during the execution ? (y/r/q)'
if(idrm.eq.0) write(4,1) 'n'
if(idrm.gt.0) write(4,1) 'y'
if(idrm.lt.0) write(4,1) 'q'
write(4,*) ' in what form store the maps ?'
write(4,*) ' s - standard '
write(4,*) ' b - bytes '
write(4,*) ' a - both (standard and bytes) '
write(4,*) ' n - don't store'
if(msav.eq.1) write(4,1) 's'
if(msav.eq.2) write(4,1) 'b'
if(msav.eq.3) write(4,1) 'a'
if(msav.eq.0) write(4,1) 'n'
write(4,*) ' take the logarithm of input data ?'
write(4,*) ' 0 - don't take logarithm'
write(4,*) ' 1 - take logarithm (once)'
write(4,*) ' 2 - take logarithm (twice) etc.'
write(4,*) ilg
write(4,*) ' the number of mineral under investigation'
write(4,*) ' (if 0 - request during the execution)'
write(4,*) miner
continue
close(4,*)
format(a1)
return
end
77
1
subroutine glider
common /llder/ ifo, isu, nm10, xmax, xmin, ymax, ymin, zmax, zmin, grid,
, madst, rdmax0, idra, lwin, idrm, msav, ilg, miner, kva
, ipix, ipog
save
character*1 r
open(4, file='leader')
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,1) r
ifo=1
if(r.eq.'f') ifo=1
if(r.eq.'u') ifo=0
if(ifo.lt.0) goto 227
read(4,*)
read(4,1) r
ipog=1
if(r.eq.'y') ipog=1
if(r.eq.'n') ipog=0
if(ipog.lt.0) goto 227
read(4,*)
read(4,1) r
isu=1
if(r.eq.'s') isu=1
if(r.eq.'f') isu=0
nm10=1
if(lisu.eq.1) goto 20
read(4,*)
read(4,*) nm10
read(4,*)
read(4,*) xmin, xmax
read(4,*)
read(4,*) ymin, ymax

```

20

77

1

30

20

2

```

read(4,*) zmin, zmax
read(4,*)
read(4,*) grld
if((isu.eq.0) goto 30
read(4,*) madst
read(4,*)
read(4,*) rdmax0
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,*) r
read(4,1) r
kva=-1
if(r.eq.'n') kva=0
if(r.eq.'1') kva=1
if(r.eq.'2') kva=2
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,1) r
ipik=-1
if(r.eq.'n') ipik=0
if(r.eq.'1') ipik=1
if(r.eq.'2') ipik=2
read(4,*)
read(4,1) r
idra=-1
if(r.eq.'y') idra=1
if(r.eq.'n') idra=0
if((isu.ne.0) goto 77
read(4,*)
read(4,1) r
lwin=0
if(r.eq.'y') lwin=1
read(4,*)
read(4,1) r
ldrm=-1
if(r.eq.'y') ldrm=1
if(r.eq.'n') ldrm=0
read(4,*)
read(4,*)
read(4,*)
read(4,*)
read(4,1) r
msav=0
if(r.eq.'s') msav=1
if(r.eq.'b') msav=2
if(r.eq.'a') msav=3
read(4,*)
read(4,*)
read(4,*)
read(4,*) llg
read(4,*)
read(4,*)
read(4,*) minor
continue
close(4,*)

```

```

1  format(a1)
   return
227  print ',' errors in file <leader>
      print ',' fix errors and restart from the beginning'
      stop
   end

   subroutine getans(ans)
   integer*4 ans,suk
   character*1 repl(4)
   equivalence(suk, repl(1))
   suk=0

      print *
      print ','          <enter> = no'
      print ','          * <enter> = yes'

      read (5,103,end=1010,err=1010) repl(4)
103  format (a1)
      if(repl(4).eq.' ') suk=0
      goto 1020

1010  rewind(5)
1020  continue
      ans=suk
      return
   end

      logical function frame( k,r ,
      real r(3,1)

***
      begin frame
      accy5= ld-2
      accy6= -ld-2
      if(frame=r(1,k).lt.-accy5.or.r(1,k).gt.(1+accy5)
         if(frame) return
      frame=r(2,k).lt.-accy5.or.r(2,k).gt.(1+accy5)
         if(frame) return
      frame=r(3,k).lt.accy6
      return
***
      end function frame
      end

```

mal14a.f

1

```

c/ mal14a
c name mal14a(r)
c mal14a/b/c - a routine to calculate the solution of the general
c linear least squares problem optionally with linear equality
c constraints. the problem is presented as m linear equations in n
c unknowns, the first m1 equations defining the constraints.
c additional right hand sides may be passed through a b entry
c which will take advantage of previous work.
c the variance-covariance matrix and an estimate of the residual
c variance is provided by the c entry.
c calculation of residuals and provision of printed output are
c are additional options.
c
c subroutine mal14a (m,n,m1,a,b,x,lw,w,lpr)
c double precision name,nme1,nme2,fm02as
c dimension a(la,1),b(1),x(1),w(1)
c dimension v(lv,1),std(1)
c dimension lw(1)
c
c m on entry: no. of equations ( including constraints ).
c n on entry: no. of unknowns.
c m1 on entry: holds the matrix of the equations; on return: holds
c the triangularized matrix and transformations; size: m*n.
c la on entry: first dimension of a.
c b on entry: holds right hand side; on return: transformed right
c hand side ( or optionally the residuals ); size: m.
c x on return: the solution; size: n.
c lw on return: column interchanges; size: n.
c w on return: transformation information; size: 2*n
c lpr on entry: specifies output and residuals options.
c lpr even: residuals; lpr odd: no residuals.
c lpr > 0: no printing; lpr < 0: printing done.
c
c restrictions: m.ge.n; 0.le.m1.le.n.
c
c reference: bjorck and golub, tech. report cs83, stanford univ.
c december 1971
c
c private variables
c logical lswitch,debug,neqml,res,meqn,rths,vflag
c data nme1/' mal14a: '//,nme2/' mal14b: '//
c
c mal14a the initial entry point
c
c rths=.false.
c name=nme1
c go to 10
c
c mal14b an entry for supplying a new right hand side to a system
c of equations solved previously, the variables m,n,m1,a,b,lw
c and w are assumed to have been unchanged.
c
c entry mal14b (b,x,lpr)
c
c b on entry: new right hand side; on return: as above.
c x on return: new solution; size: n.
c lpr as above.
c
c name=nme2
c rths=.true.
c go to 10

```

```

c entry to obtain variance-covariance matrix
c
c entry mal14c(v,lv,var,std,lpr)
c
c v on return: holds variance-covariance matrix; size: n*n.
c lv on entry: first dimension of v.
c var on return: estimate of residual variance.
c std on return: solution standard deviations; size: n.
c lpr as above.
c
c vflag=.true.
c go to 540
c
c main entry; set switches for printing and evaluating residuals.
c
c if lpr even residuals required ( res = true )
c 10 res=.false.
c if(mod(labs(lpr),2).eq.0) res=.true.
c identify special case of m = n
c meqn=.false.
c if(m-n)730,20,30
c 20 meqn=.true.
c identify special case of n = m1
c neqm1=.false.
c if(n-m1)730,40,50
c 30 neqm1=.true.
c 40 neqm1=.true.
c 50 if(m1.lt.0) go to 730
c
c vflag=.false.
c if(lpr.ge.0) go to 70
c
c print headings etc.
c
c mm=m-m1
c write(6,60) name,m1,mm,n
c 60 format('0',a8,'constraint equations = ',13/
c * 9x,'least squares equations = ',13/
c * 9x,'number of parameters = ',13)
c
c initialize and compute column scale factors
c 70 if(rths) go to 320
c nl=n+1
c ml=m1+1
c nn=n-1
c initialize column interchange indicator and scale factors
c do 90 j=1,n
c w(n+j)=0.
c 80 lw(j)=j
c calculate column scaling factors.
c do 110 i=1,m
c amx=0.
c do 90 j=1,n
c 90 amx=amax1(abs(a(i,j)),amx)
c if(amx.eq.0.) go to 110
c do 100 j=1,n
c 100 w(n+j)=amax1(w(n+j),abs(a(i,j))/amx)
c 110 continue
c do 120 j=1,n
c 120 w(n+j)=1./(w(n+j)*w(n+j))
c k=1
c
c if ml = 0 bypass first transformation step.
c if(ml.eq.0) go to 290

```

mal00650
mal00660
mal00670
mal00680
mal00690
mal00700
mal00710
mal00720
mal00730
mal00740
mal00750
mal00760
mal00770
mal00780
mal00790
mal00800
mal00810
mal00820
mal00830
mal00840
mal00850
mal00860
mal00870
mal00880
mal00890
mal00900
mal00910
mal00920
mal00930
mal00940
mal00950
mal00960
mal00970
mal00980
mal00990
mal01000
mal01010
mal01020
mal01030
mal01040
mal01050
mal01060
mal01070
mal01080
mal01090
mal01100
mal01110
mal01120
mal01130
mal01140
mal01150
mal01160
mal01170
mal01180
mal01190
mal01200
mal01210
mal01220
mal01230
mal01240
mal01250
mal01260
mal01270
mal01280

mal4a.f

```

      :
      :
      :      begin transformation step q1 on constraint equations.
      :
      :      nx=ml
      :      mx=ml
      :      mx1=mx+1
      :
      :      calculate initial partial column norms and select largest
      :      solution array used as temporary storage space for norms.
      :
      :      130 amx=0.
      :      do 140 j=k,n
      :      x(j)=(m02as(mx1-k,a(k,j),1,a(k,j),1)
      :      test=abs(x(j))*w(n+j)
      :      if(test.le.amx) go to 140
      :      j=x+1
      :      amx=test
      :      140 continue
      :
      :      calculate partial column norms using old values, and select largest
      :      150 k=k+1
      :      amx=0.
      :      do 160 j=k,n
      :      x(j)=x(j)-a(k-1,j)*a(k-1,j)
      :      test=abs(x(j))*w(n+j)
      :      if(test.le.amx) go to 160
      :      j=x+1
      :      amx=test
      :      160 continue
      :
      :      interchange columns so that col. with largest norm is used as pivot.
      :      do 180 i=1,m
      :      wx=a(i,j)
      :      a(i,j)=a(i,i)
      :      a(i,i)=wx
      :      interchange column norms.
      :      wx=x(i)
      :      x(i)=x(i)
      :      x(i)=wx
      :
      :      record column interchanges
      :      iw=iw+(j)
      :      iw(j)=i
      :      iw(i)=j
      :
      :      interchange column scale factors.
      :      wx=w(n+j)
      :      w(n+j)=w(n+i)
      :      w(n+i)=wx
      :
      :      calculate sigma, beta and transformation vector u(i) (see write
      :      up of the method).
      :
      :      190 x(k)=m02as(mx1-k,a(k,j),1,a(k,k),1)
      :      sigma=sqrt(x(k))*sign(1.,a(k,k))
      :      u(k) stored in w(k) and u(i) in a(i,k) 1.gt.x
      :      w(k)=sigma*a(k,k)
      :      beta=1./sigma*w(k)
      :      pivot element.
      :      a(k,k)=sigma
      :      k=k+1
      :      if this is the last transformation of this step branch
      :      if(k.eq.nx) go to 220
      :      apply transformation. h = scalar product (u,a(j))
      :      do 210 j=k1,n
      :      h=m02as(mx-k,a(k1,k),1,a(k1,j),1)

```

```

      h=beta*(h+w(k)*a(k,j))
      do 200 l=k1,mx
      200 a(l,j)=a(l,j)-a(l,k)*h
      210 a(l,j)=a(l,j)-w(k)*h
      go to 150
      220 if(nx.eq.n) go to 300
      :      finish of transformation.
      :      do 230 j=k1,n
      :      230 a(k,j)=(1.-beta*w(k)*w(k))*a(k,j)
      :
      :      begin elimination of first m variables in equations of condition
      :
      :      form multipliers for elimination.
      :      240 do 250 l=ml,m
      :      250 a(l,j)=a(l,j)/a(l,1)
      :      if(ml.eq.1) go to 270
      :      do 260 j=2,ml
      :      do 260 l=ml,m
      :      wx=a(l,j)-m02as(j-1,a(l,1),1,a(l,j),1)
      :      260 a(l,j)=wx/a(j,j)
      :      270 if(neqm1) go to 320
      :      carry out elimination.
      :      do 280 j=ml,n
      :      do 280 l=ml,m
      :      280 a(l,j)=a(l,j)-m02as(m1,a(l,1),1,a(l,j),1)
      :
      :      begin transformation step q2 on equations of condition.
      :
      :      k=k+1
      :      290 mx=m
      :      nx=n
      :      mx1=mx+1
      :      go to 130
      :
      :      start processing right hand side.
      :
      :      beta(k) stored in w(n+1) 1=1 to n
      :      300 do 310 j=1,n
      :      310 w(n+j)=1./w(j)*a(j,j)
      :      if(res.and.neqm1.and.not.meqn) go to 240
      :      320 iswch=.true.
      :
      :      apply transformation q1 to first m1 right hand sides.
      :      k=1
      :      nx=ml
      :      mx=ml
      :      330 a1=j+1
      :      h=m02as(mx-k,a(k1,k),1,b(k),1)
      :      h=w(n+k)*(h+w(k)*b(k))
      :      do 340 l=k1,mx
      :      340 b(l)=b(l)-a(l,k)*h
      :      b(k)=b(k)-w(k)*h
      :      if(lswch) go to 350
      :      back transform for residuals in progress.
      :      k=k+1
      :      if(k.gt.m1) go to 330
      :      go to 500
      :      350 k=k+1
      :      360 if(k-nx) 330,370,470
      :      370 if(k.lt.mx) go to 330

```

```

mal01290
mal01300
mal01310
mal01320
mal01330
mal01340
mal01350
mal01360
mal01370
mal01380
mal01390
mal01400
mal01410
mal01420
mal01430
mal01440
mal01450
mal01460
mal01470
mal01480
mal01490
mal01500
mal01510
mal01520
mal01530
mal01540
mal01550
mal01560
mal01570
mal01580
mal01590
mal01600
mal01610
mal01620
mal01630
mal01640
mal01650
mal01660
mal01670
mal01680
mal01690
mal01700
mal01710
mal01720
mal01730
mal01740
mal01750
mal01760
mal01770
mal01780
mal01790
mal01800
mal01810
mal01820
mal01830
mal01840
mal01850
mal01860
mal01870
mal01880
mal01890
mal01900
mal01910
mal01920

```

```

mal01933      h=beta*(h+w(k)*a(k,j))
mal01940      do 200 l=kl,mx
mal01950        200 a(l,j)=a(l,j)-a(l,k)*h
mal01960        210 a(k,j)=a(k,j)-w(k)*h
mal01970          go to 150
mal01980        220 if(mx.eq.n) go to 300
mal01990      c finish off transformation.
mal02000      do 230 j=kl,n
mal02010        230 a(k,j)=l.-beta*w(k)*w(k))*a(k,j)
mal02020
mal02030      c
mal02040      c begin elimination of first m1 variables in equations of condition
mal02050      c
mal02060      c form multipliers for elimination.
mal02070      do 250 l=m11,m
mal02080        250 a(l,1)=a(l,1)/a(1,1)
mal02090        if(ml.eq.1) go to 270
mal02100          do 260 j=2,m1
mal02110            do 260 l=m11,m
mal02120              wx=a(l,j)-fm02as(j-1,a(l,1),1a,a(l,j),1)
mal02130            260 a(l,j)=wx/a(f,j)
mal02140          270 if(neqml) go to 320
mal02150          c carry out elimination.
mal02160          do 280 j=m11,n
mal02170            do 280 l=m11,m
mal02180              280 a(l,j)=a(l,j)-fm02as(ml,a(l,1),1a,a(l,j),1)
mal02190
mal02200      c
mal02210      c begin transformation step q2 on equations of condition.
mal02220      c
mal02230      c k=k+1
mal02240      290 mx=m
mal02250      nx=n
mal02260      mx1=mx+1
mal02270      go to 130
mal02280
mal02290      c
mal02300      c start processing right hand side.
mal02310      c
mal02320      c beta(k) stored in w(n+1) l=1 to n
mal02330      do 310 j=1,n
mal02340        310 w(n+j)=1. / (w(j)*a(j,j))
mal02350        if(res.and.neql.and..not.meqn) go to 240
mal02360        320 lswitch=.true.
mal02370
mal02380      c
mal02390      c apply transformation q1 to first m1 right hand sides.
mal02400      k=1
mal02410      nx=m1
mal02420      mx=m1
mal02430      if(mi-1)410,380,330
mal02440        330 kl=k+1
mal02450          h=fm02as(mx-k,a(kl,k),1,b(kl),1)
mal02460          h=w(n+k)*(h+w(k)*b(k))
mal02470          do 340 l=kl,mx
mal02480            340 b(l)=b(l)-a(l,k)*h
mal02490            b(k)=b(k)-w(k)*h
mal02500          if(lswitch) go to 350
mal02510          c back transform for residuals in progress.
mal02520          k=k+1
mal02530            if(k.gt.ml) go to 330
mal02540            go to 500
mal02550          350 k=k+1
mal02560          360 if(k-nx)330,370,420
mal02570          370 if(k.lt.mx) go to 330
mal02580
mal02590      c
mal02600      c
mal02610      c
mal02620      c
mal02630      c
mal02640      c
mal02650      c
mal02660      c
mal02670      c
mal02680      c
mal02690      c
mal02700      c
mal02710      c
mal02720      c
mal02730      c
mal02740      c
mal02750      c
mal02760      c
mal02770      c
mal02780      c
mal02790      c
mal02800      c
mal02810      c
mal02820      c
mal02830      c
mal02840      c
mal02850      c
mal02860      c
mal02870      c
mal02880      c
mal02890      c
mal02900      c
mal02910      c
mal02920      c
mal02930      c
mal02940      c
mal02950      c
mal02960      c
mal02970      c
mal02980      c
mal02990      c
mal03000      c

```

ma14a.f

```

380 b(k)=b(k)*(1.-w(n+k))*w(k)*w(k)
k=k+1
if(nx.eq.n) go to 420
c
c elimination step for right hand sides.
390 do 400 i=ml,m
400 b(i)=b(i)-fm02as(ml,a(i),1),la,b(i),1)
410 if(neqn) go to 430
c
c begin applying transposition: ation q2
nx=n
mx=m
go to 360
420 if(res.and.neqn.and..not.meqn) go to 390
c
c back substitution to obtain solution which is stored temporarily
c
c in b(i) i=1 to n.
430 b(n)=b(n)/a(n,n)
if(n.eq.1) go to 450
do 440 i=1,n-1
i=n-1
wx=b(i)-fm02as(n-i,a(i),1),la,b(i),1)
440 b(i)=wx/a(i,i)
c
c finish off and compute residuals if required
c
c put the solution in its correct order and store in x(j) j=1,n
450 do 460 i=1,n
ix=ix(i)
x(ix)=b(i)
460 continue
c
c if(.not.res) go to 500
c residuals required
470 do 480 i=1,n
480 b(i)=0.
if(meqn) go to 500
do 490 i=ml,m
490 b(i)=b(i)
if(neqn) go to 500
iswch=.false.
mx=m
k=n
go to 330
500 if(vflag) go to 550
c
c print solution and residuals.
c
c if(lpr.gt.0) return
write(6,510) name
510 format('0',a8,1x,'solution')
call oa02a(x,n,10,6)
if(.not.res) go to 530
write(6,520) name
520 format('0',a8,1x,'residuals')
call oa02a(b,m,10,6)
main return point to calling program.
530 return
c
c section for obtaining variance-covariance matrix
540 if(meqn) go to 700

```

```

c
c compute estimate of residual variance
c
if(.not.res) go to 470
550 sumsq=fm02as(m,b(1),1,b(1),1)
var=sumsq/mx0(1,m-n)
if(neqn) go to 710
c
c let r be upper triangle of a then invert r
do 560 i=1,n
c save interchanges
std(i)=1-w(i,i)+.5
i=n+1-i
v(i,i)=1./a(i,i)
if(i.eq.1) go to 570
i=i-1
do 560 j=i-1,i
j=i-j
*kl=fm02as(i-j,a(j),1),la,v(j+1,i),1)
560 v(j,i)=wx/a(j,j)
c
c now form (r**-(k)trans(r**-(k)), where (k) is
c the identity matrix with first m diagonals zero
570 do 580 i=1,n
do 580 j=i,n
k=mx0(j,ml+1)
v(i,j)=fm02as(n-k+1,v(i,k),1v,v(j,k),1v)
v(i,j)=v(i,j)*var
580 v(j,i)=v(i,j)
c
c re-order rows and columns
do 630 i=1,n
590 if(i.eq.lw(i)) go to 620
k=lw(i)
do 600 j=i,n
wx=v(j,i)
v(j,i)=v(j,k)
600 v(j,k)=wx
do 610 j=1,n
wx=v(i,j)
v(i,j)=v(i,k)
610 v(i,k)=wx
lwx=lw(i)
lw(i)=lw(k)
lw(k)=lwx
go to 590
c restore interchanges
620 lw(i)=std(i)
std(i)=sqrt(v(i,i))
630 continue
c
640 if(lpr.gt.0) return
write(6,650) var
650 format('0 mal4c: residual variance estimate',e14.6/
'0 mal4c: solution standard deviations')
write(6,660) (std(j),j=1,n)
660 format(8x,e14.6)
write(6,670)
670 format('0 mal4c: variance-covariance matrix')
do 680 i=1,n
680 write(6,690) (v(i,j),j=1,n)
690 format(2x,9e14.6)
return
c
c special case m = n

```

```

mal03210
mal03220
mal03230
mal03240
mal03250
mal03260
mal03270
mal03280
mal03290
mal03300
mal03310
mal03320
mal03330
mal03340
mal03350
mal03360
mal03370
mal03380
mal03390
mal03400
mal03410
mal03420
mal03430
mal03440
mal03450
mal03460
mal03470
mal03480
mal03490
mal03500
mal03510
mal03520
mal03530
mal03540
mal03550
mal03560
mal03570
mal03580
mal03590
mal03600
mal03610
mal03620
mal03630
mal03640
mal03650
mal03660
mal03670
mal03680
mal03690
mal03700
mal03710
mal03720
mal03730
mal03740
mal03750
mal03760
mal03770
mal03780
mal03790
mal03800
mal03810
mal03820
mal03830
mal03840

```

na14a.f

```

700 var=0.
      special case ml = n
710 do 720 i=1,n
      do 720 j=1,n
720 v(i,j)=0.
      go to 640

      error conditions

730 write(6,740) name
740 format('O',a8,' error: argument restrictions violated')
      mm=n-m1
      write(6,60) name,ml,mm,n
      go to 530
      end

      name oa02a(c)
      subroutine oa02a(b,m,nc,nsfa)
      integer fnt,itrans
      dimension b(1),fnt(9),itrans(24)
      data fnt/4h(3x,1h4,2h1,1h5,2h,2h24,1h,,2h16,2h)/
      data itrans/1h1,1h2,1h3,1h4,1h5,1h6,1h7,1h8,1h9,
      . 2h10,2h11,2h12,2h13,2h14,2h15,2h16,2h17,2h18,2h19,
      . 2h20,2h21,2h22,2h23,2h24/
      nsf=max0(1,min0(nsfa,16))
      nf=nsf+8
      fnt(6)=itrans(nf)
      fnt(8)=itrans(nsf)
      m10=m
      l10=2
      10 m10=m10/10
      l10=l10+1
      if(m10.ne.0) goto 10
      x=126/(nf+110)
      fnt(4)=itrans(l10)
      fnt(2)=itrans(x)
      nsf=nsf
      20 lm=(m-1)/nc+1
      lk=(lm-1)/k+1
      lt=m-(lm-1)*nc
      itab=0
      l1=nc
      l3=nc*(k-1)
      30 if(lk-1) 90,40,50
      40 l1=lt
      l3=(lm-1)*nc
      50 do 60 l=1,l1
      l2=ltab+l
      l4=12+l3
      60 write(6,fmc) (j,b(j),j=12,14,nc)
      if(lm.eq.1.or.l1.eq.nc) goto 80
      l3=l3-nc
      l1=l1+1
      do 70 l=1,l1,nc
      l2=ltab+l
      l4=12+l3
      70 write(6,fnt) (j,b(j),j=12,14,nc)
      80 itab=ltab+nc*x
      lk=lk-1
      lm=lm-x
      write(6,999)
      999 format('O')
      goto 30
      90 return

```

oa000510

```

      end
      *
      function fm02as(n,a,la,b,lb)
      real*8 fm02as
      real a(1),b(1)
      fm02as=0
      do 1 k=1,n
      fm02as=fm02as+a(1+(k-1)*la)*b(1+(k-1)*lb)
      continue
      return
      end

```

ma103850
ma103860
ma103870
ma103880
ma103890
ma103900
ma103910
ma103920
ma103930
ma103940
ma103950
ma103960
ma103970
ma103980
ma103990

oa000050
oa000060
oa000070
oa000080
oa000090
oa000100
oa000110
oa000120
oa000130
oa000140
oa000150
oa000160
oa000170
oa000180
oa000190
oa000200
oa000210
oa000220
oa000230
oa000240
oa000250
oa000260
oa000270
oa000280
oa000290
oa000300
oa000310
oa000320
oa000330
oa000340
oa000350
oa000360
oa000370
oa000380
oa000390
oa000400
oa000410
oa000420
oa000430
oa000440
oa000450
oa000460
oa000470
oa000480
oa000490
oa000500

show.f

1

```

subroutine show(text1,ihardco)
*
* triangulation of initial points and adding of dual points
*
* flat case
*
* 'nost' - # points, 'r' - their coordinates
*
* 'ntr' - # triangles
*
* next - pointers for neighbouring triangles
*
* sites - pointers for points of the triangle
*
* radius - radius of circle for given triangle
*
* center - its center
*
* maxst - maximal number of points in the model
*
include<fgl.h>
include<device.h>
parameter (maxst=70000
*
* ,mmaxtr=2*maxst)
*
common /points/ nost,ntr
character*50 text1,filename
real r(3,maxst)
integer next(3,mmaxtr),sites(3,mmaxtr)
real radius(mmaxtr),center(2,mmaxtr)
real radv(mmaxtr),lord(mmaxtr)
*
* pepectpoltx tplahgulqcl
*
common/lala/next,sites,r
common /work1/ radius,center, radv,lord
integer systeme

common /llder/lfo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
*
* ,madst,rdmax0,ldra,iwin,ldrm,msav,ilg,miner,kva
*
* ,ipik,ipog
*
common /win/ lgld

open(1,file='rescales',status='old',form='unformatted')
open(2,file='tempor',status='unknown',form='unformatted')
open(7,file='triang')
open(4,file='leader')
call glider

do 10 il=1,maxst-10
read (1,end=1020,err=1010) r(1,il),r(2,il),r(3,il)
10 continue
print *, '===> dimension maxst too small <===>'
stop
1010 print *, ' ===> error reading <===>'
stop
1020 nost=il-1
1030 continue
print *, ' ===> nost, ' points have been read <===>'

call trlala(r,next,sites,radius,center)
call try1(text1)
call windo(r,next,sites,radius,center)
call trlala(r,next,sites,radius,center)

print *, ' triangulation of initial points is constructed'
ldrl=ldra
if(ldra.le.0) then
print *, ' draw the triangulation ?'
call getans(ldrl)
endif
if(ldrl.ne.0) then
call dsna(200,r)
endif
if(rdmax0.lt.0) then

```


show.f

2

```

4 0.0,0.8,0.0,
4 0.0,1.0,0.0,
4 0.2,0.9,0.0,
4 1.0,1.0,0.0,
4 1.0,0.8,0.0,
4 1.0,0.6,0.0,
4 1.0,0.4,0.0,
4 1.0,0.2,0.0,
4 1.0,0.0,0.0 /
real r(3,1)
common /llder/ lfo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
madst,rdmax0,ldra,iwin,ldrm,msav,ilg,miner,kva
4
integer sites(3,1)
real x(2),y(3),z(3)
real backvec(3)/0.0,0.0,0.0/
real bordovec(3)/0.3,0.0,0.3/
nm(1)=mod(1,3)+1
sca = (ncol-2)/(zmax-zmin)
call c3f(backvec)
call clear
mhr = 1
do 30 k=1,2
do 10 nt=1,ntr
z1=1
do 19 i=1,3
if(r(3,sites(mm(1),nt)).lt.zmin-9)z1=0
continue
if(z1.lt.1) then
if(k.eq.1) then
if(mod4.eq.0) then
call bgnclo
else
call bgnpol
end if
do 21 i=1,3
x(1)=r(1,sites(mm(1),nt))
x(2)=r(2,sites(mm(1),nt))
if(x(1).lt.0.or.x(1).gt.1) goto 21
if(x(2).lt.0.or.x(2).gt.1) goto 21
if(mod4.eq.0) then
call c3f(bordovec)
else
call c3f(backvec)
end if
call v2f(x)
continue
if(mod4.eq.0) then
call endclo
else
call endpol
end if
end if
goto 10
end if
if(mod4.eq.0)then
call bgnclo
else
call bgnpol
end if
do 20 l=1,3
mcolx= (sca*(r(3,sites(mm(1),nt)-zmin)))+2.0
x(1)=r(1,sites(mm(1),nt))
x(2)=r(2,sites(mm(1),nt))
if(x(1).lt.0.or.x(1).gt.1) goto 20
if(x(2).lt.0.or.x(2).gt.1) goto 20
call c3f(legend(1,mcolx))
call v2f(x)
continue
if(mod4.eq.0)then
call endclo
else
call endpol
end if
continue
return
end
10
subroutine chunks(r,mst)
parameter(ncol=13)
common /llder/ lfo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
madst,rdmax0,ldra,iwin,ldrm,msav,ilg,miner,kva
4
ipik,ipog
common /points/ nost,ntr
real legend(3,ncol)/
4 0.0,0.0,0.0,
4 0.0,0.0,0.8,
4 0.0,0.0,1.0,
4 0.0,0.6,0.6,
4 0.0,0.8,0.0,
4 0.0,1.0,0.0,
4 0.2,0.9,0.0,
4 1.0,1.0,0.0,
4 1.0,0.8,0.0,
4 1.0,0.6,0.0,
4 1.0,0.4,0.0,
4 1.0,0.2,0.0,
4 1.0,0.0,0.0/
real r(3,1)
sca = ncol/(zmax-zmin)
mcol = int((r(3,nt)-zmin)*sca)
radius=.1/sqrt(1.*nost)
do 10 nt=1,mst
call c3f(legend(1,mcol))
call circf(r(1,nt),r(2,nt),0.5*radius)
continue
return
end
10
subroutine dsna(int,r)
common /points/ nost,ntr
real r(3,1)
call v1rs
do 10 np=1,nost
x=r(1,np)
y=r(2,np)
if(x.lt.0.or.x.gt.1) goto 10
if(y.lt.0.or.y.gt.1) goto 10
11=x*255
12=255-y*255
continue
return
end
10

```

```

subroutine windo(r,next,sites,radius,center)
include<fgl.h>
include<fdevice.h>
common /popup_stuff/ menu,submenu

parameter(ncol= 13,step = 1.0/ncol,delta= 0.007)
common /win/ lgld
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
character*1 stringl*14,string2*25,string3*25
common /points/ nost,ntr
common /mccol/
common /mode/ exitmode,moda,pointmode,lcourt
common /llder/ ifo,isu,nm10,xmax,xmin,ymax,zmax,zmin,grid,
madst,rdmax0,ldra,iwin,ldrm,msav,ilg,miner,kva
,ipik,ipog
real backvec(3)/0.0,0.0,0.0/
real yelvec(3)/1.0,1.0,0.0/
real bluevec(3)/0.0,0.0,1.0/
call dtrist0,int,r,sites)
call palette
write(stringl,*)
format(f8.4)
call cmov2(1.04,0.50)
call charst(stringl,4)
write(string2,*) 'Hit the right button'
write(string3,*) % to get the menu %
call c3f(bluevec)
call rectf(1.01,0.95,1.25,0.99)
call c3f(yelvec)
call cmov2(1.02,0.96)
call charst(string3,23)
call cmov2(1.02,0.98)
call charst(string2,23)

call swapbu
return
end

subroutine processing(r,next,sites,radius,center)
#include <fgl.h>
#include <fdevice.h>
integer*2 val,mx,my,mcx,mcy
integer dev
integer do_job,ch_col
integer menuval
common /mode/ exitmode,moda,pointmode,lcourt
common /popup_stuff/ menu,submenu
character*1 string2*30,string3*30

real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)

do_job = 0
ch_col = 0
dev = qtest()
format(f8.4)

do 1000 while (dev .ne. 0)
dev = qread(val)
if(dev .eq. RIGHTM) then
menuval = dopup(menu)
if (menuval .eq. 6) then
exitmode = 1
goto 999
end if
if (menuval .eq. 5) then
moda = 1
call Drawobj(r,next,sites,radius,center)
end if
if (menuval .eq. 4) then
moda = 0

```

```

subroutine change_col(xcol,ycol,r,next,sites,radius,center)
include<fgl.h>
include<device.h>
parameter(ncol = 13,xl=1.15,yl = 0.05,six = 0.1)
common mncol
common /points/ nost,ntr
common /lidx/ lfo,lsu,nm10,xmax,xmin,ymax,ymín,zmax,zmín,grid,
& madst,rldmax0,lidx,lwin,lfrm,msav,ilg,miner,kva
& ,ipst,ipox
character*1 string1*12,string2*12
character*1 string4*12,string3*12
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
integer tgl
logical ok
integer*2 xcol,ycol

```

```

return
end

subroutine make_change(x,y,r,next,sites,radius,center)
#include<fgl.h>
#include<device.h>
parameter(ncol = 13,Delta = 0.007)
common mocol
common /points/ nost,ntr
common /mode/ exitmode,moda,pointmode, icount
common /lider/ lfo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
madst,rdmax0,ldra,iwin,ldrm,msav,ilg,miner,kva
c
c ,ipik,ipog
c
integer*2 x,y
character*1 string1*12,string2*30,string3*30,string4*30
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
integer tgl
logical ok

real x3,y3
real legend(3,ncol)/
c 0.0,0.0,0.0,
c 0.0,0.0,0.8,
c 0.0,0.0,1.0,
c 0.0,0.6,0.6,
c 0.0,0.8,0.0,
c 0.0,1.0,0.0,
c 0.2,0.9,0.0,
c 1.0,1.0,0.0,
c 1.0,0.8,0.0,
c 1.0,0.6,0.0,
c 1.0,0.4,0.0,
c 1.0,0.2,0.0,
c 1.0,0.0,0.0 /

real bluevec (3)/0.0,0.0,1.0/
real yelvec (3)/1.0,1.0,0.0/

tgl = 1

x3=(1.3/1270)*x
y3= (1.0/1000)*y
* adding
if(pointmode .eq. 1) then
z = 0
call add_point(x3,y3,z ,r,next,sites,radius,center)
end if

* copy
if(pointmode .eq.2) then
icount = icount+1
if(icount .eq. 1) then
call copy_point(x3,y3,z,r,next,sites,radius,center)
else
call add_point(x3,y3,z,r,next,sites,radius,center)
end if
end if

```

show.f

```

* removing
  if (pointmode .eq. 3) then
    call remove_point(x3,y3,r,next,sites,radius,center)
  end if

  delz = (zmax - zmin)/(ncol-2)
  call dtris(0,int,r,sites)
  call palette
  if (pointmode .eq. 1) then
    if (mccol .gt. 1) then
      write(string1,12) zmin+mccol*delz
    else
      if (mccol .eq. 1) then
        write(string1,*) ' Hole'
      else
        write(string1,*)
      end if
    end if
  end if
  if (pointmode .eq. 2) then
    write(string4,*) 'The point is chosen'
    write(string1,12) z
  end if

12  format(18.4)
    call cmov2(1.04,0.50)
    call charst(string1,12)
    call c3f(bluevec)
    call rectf(1.01,0.95,1.25,0.99)
    call c3f(yelvevec)
    write (string2,*) 'Hit right button'
    write (string3,*) 'to get menu'
    call cmov2(1.02,0.96)
    call charst(string3,23)
    call cmov2(1.02,0.98)
    call charst(string2,23)
    if (pointmode .eq. 2) then
      call cmov2(1.001,0.57)
      call c3f(blackvec)
      call charst(string4,23)
    end if

    call swappu
    return
  end

  subroutine vector(r,n1,n2,kt,1)
    parameter(ncol=13)
    common /llder/ ifo,isu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
    & madst,rdmax0,ldra,lwin,ldrm,msav,ilg,miner,kva
    & ,ipik,ipog
    & real legend(3,ncol)/
    & 0.0,0.0,0.0,
    & 0.0,0.0,0.8,
    & 0.0,0.0,1.0,
    & 0.0,0.6,0.6,
    & 0.0,0.8,0.0,
    & 0.0,1.0,0.0,
    & 0.2,0.9,0.0,
    & 1.0,1.0,0.0,
    & 1.0,0.8,0.0,
    & 1.0,0.6,0.0,
    & 1.0,0.4,0.0,
    & 1.0,0.2,0.0,
    & 1.0,0.0,0.0 /

    real link(3)/0.9,0.9,0.9/
    real blackvec(3)/0.0,0.0,0.0/
    real r(3,1)
    x1=r(1,n1)
    x2=r(2,n1)
    x3=r(3,n1)
    y1=r(1,n2)
    y2=r(2,n2)
    y3=r(3,n2)
    sca = ncol/(zmax-zmin)
    mcolx= (x3-zmin)*sca
    mcoly= (y3-zmin)*sca
    if (kt.eq.0) then
      call c3f(link)
    else
      call c3f(blackvec)
    end if
    call move(x1,x2,0.0)
    call draw(y1,y2,0.0)
    call c3f(legend(1,mcolx))
    call circf(x1,x2,.005)
    call c3f(legend(1,mcoly))
    call circf(y1,y2,.005)
    return
  end

  subroutine remove_point(x3,y3,r,next,sites,radius,center)
    !include<fgl.h>
    !include<fdevice.h>
    parameter(ncol = 13,Delta = 0.007)
    common mccol
    common /points/ nost,ntr
    common /mode/ exitmode,moda,pointmode ,icount
    common /llder/ ifo,isu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
    & madst,rdmax0,ldra,lwin,ldrm,msav,ilg,miner,kva
    & ,ipik,ipog
    & real x3,y3
    & real r(3,1),point(2)
    & integer next(3,1),sites(3,1)
    & real radius(1),center(2,1)
    & integer tgl
    & logical ok

    real blackvec (3)/0.0,0.0,0.0/
    tgl = 1

    r(1,nost+1) = x3
    r(2,nost+1) = y3
    r(3,nost+1) = zmin - 10
    rad = Delta**2

    call search(tgl,nost+1,ok,mk,r,next,sites)
    call distance(r,sites,nost+1,tgl,rad,m)
    if (m .lt. 0) return

    call frontb(.TRUE.)
    call c3f(blackvec)

```

```

call circf(r(1,sites(m,tgl)),r(2,sites(m,tgl)),Delta)
call frontb(.FALSE.)
n = sites(m,tgl)
* remove the point from the r
do 10 i = n,nost -1
do 20 k = 1,3
r(k,i) = r(k,i+1)
10 continue
nost=nost-1

call triala(r,next,sites,radius,center)

return
end

subroutine add_point(x3,y3,z,r,next,sites,radius,center)
#include<fgl.h>
#include<fdevice.h>
parameter(ncol = 13,Delta = 0.007)
common mocol
common /points/ nost,ntr
common /llder/ ifo,lsu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
madst,rdmax0,ldra,lwin,ldrm,msav,ilg,miner,kva
*
* real x3,y3
*
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
integer tgl
logical ok

real legend(3,ncol)/
* 0.0,0.0,0.0,
* 0.0,0.0,0.8,
* 0.0,0.0,1.0,
* 0.0,0.6,0.6,
* 0.0,0.8,0.0,
* 0.0,1.0,0.0,
* 0.2,0.9,0.0,
* 1.0,1.0,0.0,
* 1.0,0.8,0.0,
* 1.0,0.6,0.0,
* 1.0,0.4,0.0,
* 1.0,0.2,0.0,
* 1.0,0.0,0.0 /

tgl = 1

call frontb(.TRUE.)
call c3f(legend(1,mocol))
call circf(x3,y3,Delta)
call frontb(.FALSE.)
if(mocol.gt.0) then
if(mocol.eq.1) then
nost = nost+1

```

show.f

8

```
rad = Delta**2

call search(tgl,nost+1,ok,mk,r,next,sites)
call distance(r,sites,nost+1,tgl,rad,m)
if(m.lt. 0) return

n = sites(m,tgl)
mccol = (sca*(r(3,sites(m,tgl)-zmin)))+2.0
call frontb(.TRUE.)
call c3f(legend(1,mccol))
call clrcf(r(1,sites(m,tgl)),r(2,sites(m,tgl)),Delta)
call frontb(.FALSE.)
z = r(3,sites(m,tgl))

return
end
```

snake.f

```

      subroutine start(filename,norm,w,ipeak)
      ordering along the snake
      parameter(maxst=30000)
      parameter(maxpol=35000,maxwor=10000)
      parameter(nmin=1)
      parameter(nmin=16)
      'nst' - # to-ek, 'r' - ix koopdibaty
      maxpol - number of points in the initial file
      maxwor - dimension of working array
      parameter(maxlv=20)

      real*4 x(maxpol),y(maxpol)
      character*30 filename
      integer nnor(maxpol)
      integer work(maxpol*maxwor*3)
      integer iord(maxst*6*maxlv*6)
      real*4 r(2,maxst)
      real*4 f(nmin,maxst),ff(nmin),num(maxst)
      real*4 amem(maxpol*3*maxwor*3)
      common/work1/ amem
      common/lala/ num,nnor
      equivalence (amem(1),x(1))
      equivalence (amem(maxpol*1),y(1))
      equivalence (amem(maxpol*2+1),work(1))
      equivalence (amem(1),iord(1))
      equivalence (amem(maxst*6*maxlv*6+1),r(1,1))
      equivalence (amem(maxst*8*maxlv*6+1),f(1,1))

      common /llder/ ifo,isu,nm10,xmax,xmin,ymax,ymin,zmax,zmin,grid,
      madst,rdmax0,ldra,lwin,ldrm,msav,llg,mlner,kva
      ,ipik,ipog
      format(40a)
      open(2,file='restates',status = 'unknown',form = 'unformatted')
      open(4,file='leader')
      call glider
      ipik = ipeak
      if(ipik .eq. 1) ipog = 1
      if(ipog .eq. 0) then
        nm11=nm10
      else
        nm11=nm10*2
      endif
      if(nm11.gt.nmin) then
        print '','---> dimension nmin is too small <---'
        print '','---> put nmin = ',nm11,' <---'
        stop 18
      endif

      i1=1+int((log(real(maxst))/log(3.0))*.1)
      if (i1 .gt. maxlv) then
        print '','---> dimension maxlv is too small <---'
        print '','---> put maxlv = ',i1,' <---'
        stop
      endif

      nst=0
      if(ifo.eq.0) then
        open(1,file = filename,status='old',form = 'unformatted')
        do 10 i1=1,maxpol+1
          read (i, end=1020,err=1010) x(i1),y(i1)
          nst=nst+1
          continue
        :0
      else
        open(1,file = filename)
        do 11 i1=1,maxpol+1
          read (1,'end=1020,err=1010) x(i1),y(i1)
          nst=nst+1
          continue
        :1
      endif
      print '','---> dimension maxpol is too small <---'
      print '','---> put maxpol equal to amount of <---'
      print '','---> points in input file <---'
      stop 12
      1010 print '','---> error reading <---'
      stop 11
      1020 continue
      13 format(' ',16,' points have been read')
      if(xmax**2+xmin**2.le.1e-50) then
        xmax=-1.e50
        xmin= 1.e50
        do 786 i1=1,nst
          xmax=max(xmax,x(i1))
          xmin=min(xmin,x(i1))
          continue
        786
      endif
      if(ymax**2+ymin**2.le.1e-50) then
        ymax=-1.e50
        ymin= 1.e50
        do 787 i1=1,nst
          ymax=max(ymax,y(i1))
          ymin=min(ymin,y(i1))
          continue
        787
      endif
      compr1=1./(xmax-xmin)
      compr2=1./(ymax-ymin)
      compr=min(compr1,compr2)
      sx=(1.-(xmax-xmin)*compr)/2
      sy=(1.-(ymax-ymin)*compr)/2

      call agrid(x,y,nst,nnor,nst1,work,maxpol,maxwor)
      .f(nst1,qt,maxst) then
        print '','---> dimension maxst is too small <---'
        print '','---> put maxst = ',nst1,' <---'
        print '','---> or decrease the size of grid <---'
        stop 6
      endif
      630 i1=1,nst1
      num(i)=0
      rewind 1
      do 747 i1=1,nst
        if(ifo.eq.0) read(1) xx,yy,ff(i1),i1=1,nm11)
        if(ifo.ne.0) read(1,'xx,yy,ff(i1),i1=1,nm11)
        kk=nnor(i1)
        if(kk.lt.1) goto 747
        r(1,kk)=(r(1,kk)*num(kk)+xx)/(num(kk)+1.)
        r(2,kk)=(r(2,kk)*num(kk)+yy)/(num(kk)+1.)
        do 745 i1=1,nm11
          f(i1,kk)=(f(i1,kk)*num(kk)+ff(i1))/(num(kk)+1.)
          continue
        745
        num(kk)=num(kk)+1
      747
      nst=nst1
      print '','nst = ',nst
      do 788 i1=1,nst
        r(1,i1)=(r(1,i1)-xmin)*compr+sx

```


snake.f

2

```

r(2,1)=(r(2,1)-ymin)*compr+sy
if(isu.ne.0) then
  do 783 i=1,nml
    f(i,1)=f(i,1)*compr
  continue
endif
783
788 continue
  zmax=l.e50
  zmin=l.e50
  do 1786 i=1,nst
    do 1786 j=1,nml
      zmax=max(zmax,f(i,j))
      zmin=min(zmin,f(i,j))
    continue
  14 format(a,f8.3,a,f8.3)

  cola = (zmax-zmin)/log(w+1)
  do 1787 i=1,nst
    do 1787 j=1,nml
      f(i,j)=f(i,j)-zmin
      if(w.ge.1) then
        f(i,j)=1+w*f(i,j)/(zmax-zmin)
        f(i,j)=log(f(i,j))*cola
        zmax1=log(1+w*zmax/(zmax-zmin))*cola
        zmin1=log(1+w*zmin/(zmax-zmin))*cola
        zmax=zmax1
        zmin=zmin1
      end if
    continue
  1787
  zmax=zmax-zmin
  zmin=0

  if(norm.eq.1) f(i,j)=f(i,j)/(zmax-zmin)
  continue
  zmax=zmax-zmin
  zmin=0

  call snake(r,lord,nst,maxlv,maxst)

  do 30 i=1,nst
    write(2) (r(j,lord(i)),j=1,2), (f(i,lord(i)),l=1,nml)
    print *, (r(j,lord(i)),j=1,2), (f(i,lord(i)),l=1,nml)
  30 continue

  call plider,' snake '
  close(1)
  close(2)
  close(4)
  if (lnd.eq.1) then
    endif
  return
end

subroutine agrid(x,y,nst,nnor,nstl,w,maxpol,maxwor)
integer w(1),nnor(1)
real x(1),y(1)
m1=1
m2=m1+maxpol
m3=m2+maxvor
m4=m3+maxwor
call agril(x,y,nst,nnor,nstl,w(m1),w(m2),w(m3),w(m4),maxwor)
return
end

end

subroutine agril(x,y,nst,nnor,nstl,lord2,npo,y1,maxvor)
common /lider/ lfo,isu,nml0,xmax,xmin,ymax,ymin,zmax,zmin,grid,
6 madst,rdrmax0,ldra,iwin,ldrm,msav,ilq,miner,kva
6 ,ipik,ipog
real*4 x(1),y(1)
integer lord(1),nnor(1),npo(1),lor2(1)
real*4 y1(1)

do 737 ki=1,nst
  nnor(ki)=0
  print *, 'grid = ', grid
  dd=max((xmax-xmin),(ymax-ymin))/grid
  call kb07a(x,nst,lord)
  do 773 ip5=1,nst
    il=ip5
    if(x(ip5).ge.xmin) goto 774
    continue
  773
  774
  stx=xmin
  kx=0
  lcheck=0
  if(stx.gt.xmax) goto 100
  stx=stx+dd
  iy=0
  if(lcheck.gt.0) goto 100
  do 93 i=1,nst
    i2=i
    il=lord(i)
    if(x(il).gt.stx) goto 35
    iy=iy+1
    npo(iy)=il
    y1(iy)=y(il)
  93
  continue
  lcheck=1
  35
  if(iy.gt.maxvor) then
    print *, '====> dimension maxvor is too small <===='
    print *, '====> increase maxvor'
    print *, '====> or decrease the size of grid <===='
    stop
  endif
  if(iy.eq.0) goto 5
  il=i2
  call kb07a(y1,iy,lord2)
  sty=max(ymin,y1(i))+dd
  kx=kx+1
  do 43 i=1,iy
    if(y1(i).lt.ymin) goto 43
    if(y1(i).gt.sty) then
      if(sty.gt.ymax) goto 5
      sty=sty+dd
      if(y1(i).gt.sty) goto 782
      kx=kx+1
    endif
    nnor(npo(lord2(i)))=kx
  43
  continue
  goto 5
  100
  continue
  nstl=kx
  return
end

```

```

... end subroutine
*
subroutine split (lv, corner, nn, lord, mord, maxst, maxlv, x, iof)
include (param)
logical test
integer iadd(3), ibf(maxst, 2)
*, corner(4, maxlv), nn(2, maxlv), iord(maxst, 2), mord(maxst, 2)
real x(2, maxst)
begin
n1=nn(1, lv-1)
n2=nn(2, lv-1)
print *, lv n1 n2
*, print *, lv, n1, n2
if (n1.eq.n2) return
k=corner(1)+corner(1, lv-1), lv-1)
*, tip delehiq
dx=x(1, lord(n2, 1))-x(1, lord(n1, 1))
dy=x(2, lord(n2, 2))-x(2, lord(n1, 2))
if (dy.gt.dy) then
ityp=1
else
ityp=2
endif
test=ityp.eq.1.and.(k.eq.3.or.x.eq.4)
test=test.or.ityp.eq.2.and.(k.eq.2.or.k.eq.3)
*, pointers
corner(1, lv)=1
corner(2, lv)=k
corner(4, lv)=k
if (ityp.eq.1) then
corner(3, lv)=2*mod(k, 2)-1+k
else
corner(3, lv)=5-k
endif
*, split
ien=(n2-n1)/3+1
nn(1, lv)=n1
nn(2, lv)=ien+n1-1
m1=n1
do 1 i=1, 3
ladd(i)=m1
if (m1.gt.n2) goto 1
m2=min(n2, m1+ien-1)
if (test) then
*, reverse
l1=max(n2-1*ien+1, n1)
else
l1=m1
endif
do 2 j=m1, m2
jj=l1+j-m1
l1=lord(jj, ityp)
ibf(j, ityp)=l1
mord(l1, ityp)=j
2 continue
m1=m2+1
1 continue
*,
ityp=3-ityp
do 3 j=n1, n2
l1=lord(j, jtyp)
jj=mord(l1, ityp)
kk=(jj-n1)/len+1
ibf (ladd(kk), jtyp)=l1
end
end

```

snake.f

```

mord(1,1,1,typ)=ladd(kk)
ladd(kk)=ladd(kk)+1
  continue
do 33 j=n1,n2
do 33 i=1, 2
  lord(i,1)=lbf(i,1)
3  continue

  return
** end subroutine
end

subroutine snasna(n,corner,nx,lord,mord,maxst,maxiv)
include param
integer corner(4,maxiv),nn(2,maxiv),lord(maxst,2),mord(maxst,2)
logical direct
begin snasna

  m=log(float(n))/log(2.)*1
  n=2**m
  mm=m
  lord(1,2)=lord(n,1)
  lord(2,2)=lord(m/2,1)
  j=2
  direct=.true.
  lstep=m/4
  continue
  lstep2=lstep*2
  if(direct) then
do 10 i=lstep,n-1,lstep2
  if(mod(i,m).eq.0) print *, 'error'
  j=j+1
  lord(i,2)=lord(i,1)
10 continue
  direct=.false.
  else
  xx=(mm+lstep-n+1)/lstep2
  print *, 'kk=',kk
  nearn=mm-kk*lstep2-lstep
do 20 i=nearn,lstep,(-lstep2)
  if(mod(i,m).eq.0) print *, 'error2', 'i=',i,m
  j=j+1
  lord(i,2)=lord(i,1)
20 continue
  direct=.true.
  endif
  m=lstep
  lstep=lstep/2
  if(lstep.ge.1) goto 1
  open(7,file='test')
do 39 i=1, nst
  write(7) i,lord(i,2)
39 continue
  close(7)
  return
** end subroutine snasna
end

```

triang.f

1

```

subroutine trian(iharco)
*
* triangulation of initial points and adding of dual points
* flat case
* 'nst' - # points, 'r' - their coordinates
* 'ntr' - # triangles
* next - pointers for neighbouring triangles
* sites - pointers for points of the triangle
* radius - radius of circle for given triangle
* center - its center
* maxst - maximal number of points in the model
* parameter(maxst=70000
* ,mmaxtr=2*maxst)
* common /points/ nst,ntr
real r(3,maxst)
integer next(3,mmaxtr),sites(3,mmaxtr)
common/ala/next,sites,r
real radius(mmaxtr),center(2,mmaxtr)
real radv(mmaxtr),lord(mmaxtr)
common /vorx1/ radius,center, radv, lord
integer systeme

common /llder/ lfo,lsu,nml0,xmax,xmin,ymax,ymin,ldrm,msav,liq,miner,kva
*
* madst,rdmax0,ldra,iwin,ldrm,msav,liq,miner,kva
* ,lptk,lpoq
open(1,file='rescales',status='old',form='unformatted')
open(2,file='tempor',status='unknown',form='unformatted')
open(7,file='t1lang')
open(4,file='leader')
open(8,file='text')
call glider
do 10 i1=1,maxst-10
read (1,end=1020,err=1010) r(1,i1),r(2,i1),r(3,i1)
10 continue
print *, '====> dimension maxst too small <===='
stop
1010 print *, '====> error reading <===='
1020 write(8,*) 'error reading'
stop
1020 nst=i1-1
1030 continue
write(8,*) '40 30%'
print *, '====> nst, # points have been read <===='
write(8,13) nst, # points have been read'
13 format(' ',i5,a)

call trian(r,next,sites,radius,center)
call chunks(r,nst)

print *, 'triangulation of initial points is constructed'
write(8,*) 'triangulation of initial points is constructed'
ldra=ldra
if(ldra.le.0.3) then
print *, '% draw the triangulation'
call getans(ldra)
endif
if(rdrax0.lt.0) then
print *, 'radius of confidence ?'
read *,rdmax0
endif
rdmax=rdmax0**2
definition of radius of confidence

```

```

if(rdmax.le.1e-50) then
call detrad(rdmax,radius,radv,lord)
rdmax0=sqrt(rdmax)
endif
ind5 = 0
call output(r,next,sites,radius,lord,iharco)
close(8)
call plidtr(' triang ')
print *, '====> end of job <===='
if (ind .eq.1) call try1("text")
ind5 = 1
close(7)
close(1)
close(2)
return
end

subroutine trian(r,next,sites,radius,center)
common /points/ nst,ntr
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
integer tgl
logical ok

nsto0=nst
call inittr(r,next,sites,radius,center)

tgl=1
ndp=0
do 20 np=2,nsto0
if(nst.ge.1000.and.mod(np,1000).eq.0) then
print *, '====> np, # points have been included <===='
end if
bkl=itx to-ku 'np' b tpiahgulqcl'. dliq ltogo
hajti tp-k 'tgl', b kotopom levit 'np'
call search(tgl,np,ok,mk,r,next,sites)
if(ok) then
c delatx baplehtpl-eckoe podpazdelehle
call bar(tgl,np,mk,r,next,sites,radius,center)
pepectpoltx tpiahgulqcl'
call recon(tgl,r,next,sites,radius,center)
else
ndp=ndp+1
end if
20 continue
if(ndp.ne.0) then
print *, '====> ndp, # points has not been included <===='
print *, '====> check, if you execute program <===='
print *, '====> snake properly'
write(8,*) 'check, if you execute program'
write(8,*) 'snake properly'
endif
return
end

subroutine inittr(r,next,sites,radius,center)
*
* ihicilizaciq
* papametry okpuvhoct1, oplcahnoj bokpug model1
* rbnd - radiuc, nbnbp - kol-bo to-ek ha okpuvhoct1

```

triang.f

```

parameter(nbndp=11,rbnd=10.)
integer tgl
common /points/ nst,ntr
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
save

p=atan(1.)+8
ntr=0
nst=0
do 10 j0=1,nbndp
  ntr=ntr+1
  nst=nst+1
  r(1,nst)=rbnd*cos(p*j0/nbndp)+.5
  r(2,nst)=rbnd*sin(p*j0/nbndp)+.5
  r(3,nst)=0.0
  sites(1,j0)=nst
  sites(2,j0)=nst+mod(j0,nbndp)+1
  sites(3,j0)=1
  next(1,j0)=mod(j0,nbndp)+1
  next(2,j0)=mod(j0+nbndp-2,nbndp)+1
  next(3,j0)=0
10 continue
do 20 ijk=1,nbndp
  call circle(ijk,r,sites,radius,center)
20 continue
return
end

subroutine bar(tr1,st,mk,r,next,sites,radius,center)
to-ka 'st' pomejstcq b tp-k 'tr1' ppl pome)1
bapicehtpi-eckogo podpazdelehq
common /points/ nst,ntr
real r(3,1)
integer next(3,1),sites(3,1)
real radius(1),center(2,1)
integer np,tr1,tr2,tr3,tra,trb,st,st1,st2,st3
save

if(mk.gt.0) then
  n0=next(mk,tr1)
  m0=mark(n0,tr1,next)
  endif
  tr2=next(2,tr1)
  tr3=next(3,tr1)
  -2=mark(tr2,tr1,next)
  -3=mark(tr3,tr1,next)
  'tra' 1 'trb' - dba robyx tpeugolxhika
  tra=ntr+1
  trb=ntr+2
  ntr=ntr+2
  if(tr2.ne.0) next(m2,tr2)=tra
  if(tr3.ne.0) next(m3,tr3)=trb
  next(2,tr1)=tra
  next(3,tr1)=trb
  next(1,tra)=tr2
  next(2,tra)=trb
  next(3,tra)=tr1
  next(1,trb)=tr3

```

```

next(2,trb)=tr1
next(3,trb)=tra
st1=sites(1,tr1)
st2=sites(2,tr1)
st3=sites(3,tr1)

sites(1,tr1)=st
sites(1,tra)=st
sites(2,tra)=st3
sites(3,tra)=st1
sites(1,trb)=st
sites(2,trb)=st1
sites(3,trb)=st2

*film

if(mk.gt.0) then
  n1=next(m0,n0)
  if(next(1,n1).ne.n0) stop 33
  call flip(n1,1,next,sites,r)
  call circle(n0,r,sites,radius,center)
endif
return
end

subroutine search(tgl,np,ok,mk,r,next,sites)
polck tpeugolxhika b kotopyj popadaet to-ka np
parameter (e=1e-12
           ,el=1e-10)
integer tgl,m,np,prev,ndp/0/
common /points/ nst,ntr
real r(3,1)
integer next(3,1),sites(3,1)
logical ok
save
mpr=10
mk=0
a3=volum(r(1,np),r(1,sites(1,tgl)),r(1,sites(2,tgl)))
a1=volum(r(1,np),r(1,sites(2,tgl)),r(1,sites(3,tgl)))
a2=volum(r(1,np),r(1,sites(3,tgl)),r(1,sites(1,tgl)))
if(a1.ge.0.and.a2.ge.0.and.a3.ge.0) then
  v1=abs(a1)
  v2=abs(a2)
  v3=abs(a3)
  if(v1.le.e.or.v2.le.e.or.v3.le.e) then
    if(v1.le.e.and.v2.le.e.and.v3.le.e) then
      ok=.false.
      return
    endif
    if(v1.le.e) then
      if(v2.le.e) then
        ok=.false.
        ndp=ndp+1
      else
        if(v3.le.e) then
          ok=.false.
          ndp=ndp+1
        else
          mk=1
          ok=.true.
          end if
        endif
      endif
    endif
  endif

```

3

• • • •

```

integer tgl,nxt,ngb,stp
logical critter
save

nxt=tbl
stp=0
1010 ngb=next(1,nxt)
if(ngb.ne.0) then
  if((sites(1,nxt).ne.sites(1,tgl)) then
    print *, '==== "next" and "sites" are in disaccord <===='
    print *, '==== fatal error <===='
    write(8,*)'fatal error'
    stop 41
  end if
  call circle(ngb,r,sites,radius,center)
  call flip(nxt,1,next,sites,r)
  goto 1010
end if

call circle(nxt,r,sites,radius,center)
nxt=next(2,nxt)
stp=stp+1
if(stp.gt.3*stp) then
  print *, '==== can't construct triangulation <===='
  write(8,*)'fatal error'
  return
end if

if(nxt.ne.tgl) goto 1010
return
end

logical function critter(tg,st,r,radius,center)

pobepxa kpugobogo kpitep
popadaet li to-ka st b okpuvhoctx, opicahhu' bokpug tp-ka tg ?
ecli pepadaet, to critter = .false.
real r(3,1)
real radius(1),center(2,1)
integer tg,st
save
if(tg.eq.0) stop 44
if(scal(r(1,st),center(1,tg)).lt.radius(tg)) then
  critter=.false.
else
  critter=.true.
end if
return
end

subroutine flip(tga,mal,next,sites,r)
common /points/ nst,ntr
integer next(3,1),sites(3,1)
integer tga,tgb,shl
real r(3,1)
save
shi (n)=mod(m,3)+1
-a2=shl(mal)
-a3=shl(ma2)
tgb=next(mal,tga)
if(tga.eq.0) stop 45

mb1=mark(tgb,tga,next)
mb2=shl(mb1)
mb3=shl(mb2)

*film
c
c call vector(r,sites(ma2,tga),sites(ma3,tga),5,4)
c if(nst.lt.5) call sleep(1)
c call vector(r,sites(ma2,tga),sites(ma3,tga),4,13)
c call vector(r,sites(ma1,tga),sites(mbl,tgb),5,4)
c if(nst.lt.5) call sleep(1)
c call vector(r,sites(mal,tga),sites(mbl,tgb),11,7)
c call vector(r,sites(mal,tga),sites(ma2,tga),11,7)
c call vector(r,sites(ma3,tga),sites(mal,tga),11,7)
c call vector(r,sites(ma2,tgb),sites(mbl,tgb),11,7)
c call vector(r,sites(ma3,tgb),sites(mbl,tgb),11,7)
c if(nst.lt.5) call sleep(1)
*film
if(next(ma2,tga).ne.0) then
  next(mark(next(ma2,tga),tga,next),next(ma2,tga))-tgb
end if
if(next(ma2,tgb).ne.0) then
  next(mark(next(ma2,tgb),tgb,next),next(ma2,tgb))-tga
end if
next(mal,tga)=next(ma2,tgb)
next(mal,tgb)=next(ma3,tgb)
next(ma2,tgb)=next(ma2,tga)
next(ma2,tga)=tgb
next(ma3,tgb)=tga
sites(ma3,tga)=sites(mbl,tgb)
sites(ma3,tgb)=sites(ma2,tgb)
sites(mal,tgb)=sites(mal,tga)
sites(ma2,tgb)=sites(ma3,tga)
return
end

subroutine circle(tgl,r,sites,radius,center)
*
* by-iclg'tcq koopdihaty cehtpa i padluc okpuvhocti,
* opicahhoj bokpug tp-ka tgl
real r(3,1)
integer sites(3,1)
real radius(1),center(2,1)
integer tgl,st1,st2,st3
real*8 a(2,2),b(2,2),det,pp
save
pp(x)=dprod(x,x)

st1=sites(1,tgl)
st2=sites(2,tgl)
st3=sites(3,tgl)
a(1,1)=r(1,st2)-r(1,st1)
a(1,2)=r(2,st2)-r(2,st1)
a(2,1)=r(1,st3)-r(1,st1)
a(2,2)=r(2,st3)-r(2,st1)
b(1,1)=pp(r(1,st2))*pp(r(2,st2))-pp(r(1,st1))-pp(r(2,st1))/2
b(1,2)=pp(r(1,st3))*pp(r(2,st3))-pp(r(1,st1))-pp(r(2,st1))/2
det=a(1,1)*a(2,2)-a(2,1)*a(1,2)
center(1,tgl)=-(a(2,2)*b(1,1)-a(1,2)*b(2,1))/det
center(2,tgl)=-(a(1,1)*b(2,1)-a(2,1)*b(1,1))/det
radius(tgl)=pp(r(1,st1)-center(1,tgl))
+pp(r(2,st1)-center(2,tgl))

return
end

```

```

real function scal(x,y)
  scal = (x,y)
  real x(2),y(2)
  scal=dprod(x(1)-y(1),x(1)-y(1))+dprod(x(2)-y(2),x(2)-y(2))
  return
end

real function volum(x,y,z)
  plotadx co zhatom
  volume = {y-x,z-x}
  real x(2),y(2),z(2)
  volum=dprod(y(1)-x(1),z(2)-x(2))-dprod(y(2)-x(2),z(1)-x(1))
  return
end

integer function marx(tr1,tr2,next)
integer tr1,tr2
integer next(3,1)

```

```

opdeletele oplenactl b tp-xe ctrl po tp-xu ctrl
if(ctrl.eq.0) then

```

```
mark=0
return
end if
do i0 mark=i,3
  if(next(mark,ctrl).eq.ctrl) return
  continue
  print ',' , '==> "next" is in disaccord
  print ',' , '==> fatal error
  write(i0,'fatal error',
stop $i
end
end
```

subroutine output (r, next, sites, radius, lord, lhardco)

bybyod dlyq blzualizaci: i lntceppolqcll

common / studied / not

```
real = (3,1)
```

Integer next(3, 1), sizes(3, 1)

```
real rad; vs(1), center(2.);
```

logical frame

20160727 15:00:00

(7) 2017 2018

```
common /io, isu, nm10, xmax, xmin, ymax, ymin, zmax, zmin, grid,  
      madst, rmax0, ldra, twin, ldrn, msav, flg, m1ner, kva  
      , fptk, p0g
```

Save

0-6.220

[illegible]

2025-01-01

15-00000-1

11/11/11

100

100

100

44-38861-741

10-12-55/200

200 (1222) P20

0
6
7
8
9

7220 (1,0) 211

10-10-12-1, 152

12(12,12).

$$Z := \{2, 32\}.$$

```

write (7,*) r(1,j2),r(2,j2),r(3,j2)
else
write(7,*)r1,r1,r(3,j2)
end if
else
write(7,*)r1,r1,r(3,j2)
end if
continue
do 710 i3=1,ntr7
j2=lord(i3)
write (7,*) sites(1,j2),sites(2,j2),sites(3,j2),
continue
if(modst.eq.0.and.isu.eq.1) return
write (2) ntr,nst
write (2) ((sites(i0,x0),i0=1,3),x0=1,ntr)
write (2) ((next(i0,x0),i0=1,3),x0=1,ntr)
if(hardco.eq.1) then
call hardcopy(r,sites,filename)
end if
return
end

subroutine detrad(rdmax,radius,radv,lord)
common /points/ nst,ntr
real radius(1)
real radv(1),lord(1)
nnt=0
do 30 i2=1,ntr
if(radius(i2).gt.5.) goto 30
radv(i2)=radius(i2)
nnt=nnt+1
continue
call kb07a(radv,nnt,lord)
definition of radius of confidence
rr=0
nt=nnt*.9
do 10 ktr=1,nt
rr=rr+radv(ktr)
continue
rdmax=rr/nt*10
print *, 'radius of confidence =',sqrt(rdmax)
return
end

subroutine hardcopy(r,sites)
common /points/ nst,ntr
real r(3,1)
integer sites(3,1)
real x(3),y(3)
mm(1)=mod(1,3)+1
open(10,file='hardtmp')
write(10,*) 'limits 0 1 0 1'
write(10,*) 'lweight 3'
write(10,*) 'ptype 10 3'
write(10,*) 'erase'
do 10 nt=1,ntr
do 30 i=1,3
mm(i),nnt)
m2=sites(
mm(i+1),nnt,
if(mm(i).m2) goto 30
if(r(1,m1).lt.0.or.r(1,m1).gt.1) goto 30
if(r(2,m1).lt.0.or.r(2,m1).gt.1) goto 30
if(r(3,m1).lt.0.or.r(3,m1).gt.1) goto 30

```


triang.f

```

    if (r(2,m2).lt.0.or.r(2,m2).gt.1) goto 30
    write(10,*) 'relocate', r(2,m1), r(1,m1)
    write(10,*) 'draw', r(2,m2), r(1,m2)
30  continue
10  continue
    close(10)
    return
end

    dist = dist + (r(2,np) - r(2,sites(k,tgl))) ** 2
    if (k.eq.1) then
        dmin=dist
        kmin=1
    else if (dist.lt.dmin) then
        kmin=k
        dmin=dist
    endif

40  continue
    if (dmin.gt.rad) kmin=-1
    return
end

```

```

    logical function bad(n,rmin,radius,center)
    parameter (rvmin=1e-8,rvmax=0.7)
    logical cirin
    real radius(1),center(2,1)
    ** begin
        bad=radius(n).lt.rmin
        if (bad) return
        bad=radius(n).ge.rvmax
        if (bad) return
        if (bad) return
        circle inside?
        bad=.not.cirin(n,circle)
        return
    ** end function
end

```

```

    logical function cirin( nt,circle )
    real circle(2,1)
    ** begin cirin
        cirin=abs(circle(1,nt)-.5).lt..6
        cirin=cirin.and.(abs(circle(2,nt)-.5).lt..6)
        return
    ** end function cirin
end

```

```

    logical function border(n,r,sites,radius)
    parameter (rvmin=1e-8,rvmax=0.7)
    logical frame
    integer sites(3,1)
    real r(3,1),radius(1)
    ** begin
        border=radius(n).ge.rvmax
        if (border) return
        ns1=sites(1,n)
        border=frame(ns1,r)
        if (border) return
        ns2=sites(2,n)
        border=frame(ns2,r)
        if (border) return
        ns3=sites(3,n)
        border=frame(ns3,r)
        return
    ** end function
end

```

```

    subroutine distance(r,sites,np,tgl,rad,xmin)
    real r(3,1),rmin
    integer sites(3,1)
    integer tgl
    do 40 k = 1,3
        dist = (r(1,np) - r(1,sites(k,tgl))) ** 2
    enddo

```

path.h

1

'define PATH "/usr/people/lila/FILM/"
'define PLACE "< triang 4"

event.h 1

```

/*
 * event.h
 * External interface and defines to input-queue event handling
 * routines.
 * Written by Wade Olsen for Silicon Graphics, Inc.
 */

/*
 * The event handler understands two kinds of things; events and
 * updates. Events are reactions to things occurring in the input
 * queue. Updates are functions that should be called whenever there
 * is nothing waiting in the input queue, and may be active or
 * inactive. If there are no active updates and nothing in the input
 * queue, then event() will block, using up no CPU time.
 *
 * add_event is used to look for events. The first three arguments
 * are used to identify which event to look for. The first argument
 * is the window (gid) the event must happen in; if this value is ANY
 * then any window will do. The second argument is the device to look
 * for (e.g. RIGHTMOUSE or REDRAW or KEYBD, etc). Again, if it is
 * ANY then any device will match. The third argument is the value
 * the device must generate (e.g. DOWN or UP); ANY means all values
 * match. The last two arguments are what should be done when an
 * event is generated. The fourth argument is a function to be
 * called, and the fifth is an argument that should be supplied to the
 * function. In addition, the value generated by the device will also
 * be passed to the function when it is called.
 *
 * For example,
 *   add_event(widget(), RIGHTMOUSE, DOWN, dopup, my_menus);
 *   qdevice(RIGHTMOUSE);
 * will make a pop-up menu appear when the right mousebutton goes
 * down. Note that you must do the qdevice() call yourself.
 */
void add_event(int, int, int, void (*fn)(void *, int), char *) ;

/*
 * An update is like an event, only simpler. The first argument is a
 * pointer to an integer flag specifying whether or not this update
 * function is active. The second is a function to be called when it
 * is active, and the last is an argument to be supplied to the
 * function.
 */
void add_update(int *, void (*fn)(void *), char *) ;

/*
 * Finally, when all updates and events have been added, repeatedly
 * call event() to handle them -- something like
 *
 *   while (quitflag == FALSE) event();
 *
 * You should have previously added an event that sets quitflag to
 * TRUE, of course.
 */
void event(void) ;

/*
 * These are some useful defines for the possible values buttons
 * can generate.
 */
#define ANY      -1
#define UP       0
#define DOWN    1

```

```

/*
 * And a few external variables you might find useful
 */
extern int context, state, device;

```

```

#endif
#ifndef PI
#define PI 3.14159265358979323846 /* from math.h */
#endif
#define PNL_RAD_TO_DEG (0.3183098861877067154*180.0)
#define PNL_TABLE_SIZE 4096 /* max number of panels + actuators */

```

```

#define MAXSGIDDEVICE 20000
#define MAXSGIDDEVICE
#endif
#define PNL_DEVICE_OFFSET (MAXSGIDDEVICE+1)
#define PNL_TOKEN_OFFSET (PNL_DEVICE_OFFSET+0)

#define PNL_PIXELS_PER_UNIT 50.0 /* used to size the panel */
#define PNL_CHAR_THRESHOLD 35.0 /* ppm below this, draw chars as rects */

```

```

#define PNL_CHAR_PIXEL_HEIGHT 10
#define PNL_CHAR_PIXEL_WIDTH 10

```

```

endif IRIS_NEWS
12 define PNL_SCREEN_BORDER
19 define PNL_TITLE_BAR_HEIGHT
false
6 define PNL_SCREEN_BORDER
18 define PNL_TITLE_BAR_HEIGHT
endif

```

/* actuator dimensions */

```

define PNL_SLIDER_HEIGHT (6.0-PNL_DIM_1) /* w/o label */
define PNL_SLIDER_WIDTH (1.0-PNL_DIM_1)
define PNL_SLIDER_BAR_HEIGHT (0.05*PNL_SLIDER_HEIGHT)
define PNL_DIFFERENTIAL_FACTOR 0.1
define PNL_FINE_CONTROL_FACTOR 0.05
define PNL_BUTTON_EDGE (0.5-0.5*PNL_DIM_1)
define PNL_BUTTON_WIDTH (2.0-PNL_DIM_1)

```

```

define PNL_TYPEN_MAX_STRING 256
define PNL_TYPEN_MAX_STRING 4096
define PNL_TYPEOUT_BUF$12 4096
define PNL_TYPEOUT_COLUMNS PNL_TYPEN_LENGTH
define PNL_TYPEOUT_LINES 4
define PNL_TYPEOUT_MAX_COLUMNS 256
define PNL_SCROLLBAR_WIDTH (2.0*PNL_DIM_1)

```

```

#define PNL_PUCK_EDGE (4.0-PNL_DIM_1)
#define PNL_PUCK_SIZE (1.0-PNL_DIM_1)
#define PNL_FLOATING_PUCK_EDGE (1.0-PNL_DIM_1)
#define PNL_FLOATING_PUCK_SENS 10.0 /* units per full range output */
#define PNL_METER_HEIGHT (2.0-PNL_DIM_1)
#define PNL_METER_WIDTH (3.0-PNL_DIM_1)
#define PNL_METER_LINEWDTH 4 /* pixels for meter pointer and bar line */
#define PNL_STRIP_HEIGHT (2.0-PNL_DIM_1)
#define PNL_STRIP_WIDTH (4.0-PNL_DIM_1)

```

panel.h

```

#define PNL_STRIP_LINEWIDTH 2
#define PNL_STRIP_CHART_NPTS 200
#define PNL_SLIDEROID_HEIGHT (1.5-PNL_DIM_1)
#define PNL_SLIDEROID_WIDTH (2.0-PNL_DIM_1)
#define PNL_DIAL_EDGE (1.0-PNL_DIM_1)
#define PNL_DIAL_WINDS 0.88 /* revs per full range output */
#define PNL_MULTISLIDER_DIVISIONS 5
#define PNL_MENU_WIDTH PNL_WIDE_BUTTON_WIDTH
#define PNL_MENU_TITLE_HEIGHT PNL_WIDE_BUTTON_HEIGHT
#define PNL_ICON_WIDTH (2.0-PNL_DIM_1)
#define PNL_ICON_HEIGHT (0.5-PNL_DIM_2)
#define PNL_SCROLL_WIDTH (4.0-PNL_DIM_1)
#define PNL_SCROLL_HEIGHT (6.0-PNL_DIM_1)

/* Actuator types */
#define PNL_MAXACT 0x7fff /* user types start here+1 */
#define PNL_USER_OFFSET 0x8000 /* first user actuator */

#define PNL_SLIDER 0
#define PNL_VSLIDER 1
#define PNL_HSLIDER 2
#define PNL_FILLED_SLIDER 3
#define PNL_FILLED_VSLIDER 4
#define PNL_FILLED_HSLIDER 5
#define PNL_DVSLIDER 310
#define PNL_DHSLIDER 311
#define PNL_TOGGLE_BUTTON 312
#define PNL_RADIO_BUTTON 313
#define PNL_WIDE_BUTTON 314
#define PNL_SHADOW_BUTTON 315
#define PNL_LEFT_ARROW_BUTTON 316
#define PNL_RIGHT_ARROW_BUTTON 317
#define PNL_UP_ARROW_BUTTON 318
#define PNL_DOWN_ARROW_BUTTON 319
#define PNL_LEFT_DOUBLE_ARROW_BUTTON 320
#define PNL_RIGHT_DOUBLE_ARROW_BUTTON 321
#define PNL_UP_DOUBLE_ARROW_BUTTON 322
#define PNL_DOWN_DOUBLE_ARROW_BUTTON 323
#define PNL_TYPEIN 20
#define PNL_LABEL 21
#define PNL_TYPEOUT 30
#define PNL_METER 40
#define PNL_ANALOG_METER 41
#define PNL_ANALOG_BAR 42
#define PNL_STRIP_CHART 43
#define PNL_SCALE_CHART 44
#define PNL_PUCK 50
#define PNL_FLOATING_PUCK 51
#define PNL_RUBBER_PUCK 52
#define PNL_SLIDEROID 60
#define PNL_PALETTE 70
#define PNL_DIAL 80
#define PNL_MULTISLIDER 90
#define PNL_VMULTISLIDER 91
#define PNL_HMULTISLIDER 92
#define PNL_MULTISLIDER_BAR 96
#define PNL_MULTISLIDER_OPEN_BAR 97
#define PNL_VMULTISLIDER_BAR 98
#define PNL_VMULTISLIDER_OPEN_BAR 99
#define PNL_HMULTISLIDER_BAR 99
#define PNL_HMULTISLIDER_OPEN_BAR 100
#define PNL_MENU 100

#define PNL_MENU_ITEM 110
#define PNL_ICON 120
#define PNL_ICON_MENU 130
#define PNL_SUB_MENU 132
#define PNL_FRAME 140
#define PNL_CYCLE 150
#define PNL_SCROLL 160
#define PNL_MOUSE 1000

/* slider modes */
#define PNL_SM_NORMAL 0x0
#define PNL_SM_DIFFERENTIAL 0x1
#define PNL_SM_FINE_CONTROL 0x2
#define PNL_SM_NOSNAP 0x4

/* multislider modes */
#define PNL_MSM_FREE 0x00
#define PNL_MSM_ORDERED 0x01
#define PNL_MSM_CONSTRAINED 0x02
#define PNL_MSM_ADD 0x04
#define PNL_MSM_DELETE 0x08

/* icon modes */
#define PNL_IM_STORED 0x01
#define PNL_IM_OPEN 0x02

/* typeout modes */
#define PNL_TOM_NORMAL 0x00
#define PNL_TOM_NOCURSOR 0x01
#define PNL_TOM_NOREGION 0x02

/* frame modes */
#define PNL_FM_FREE 0x00
#define PNL_FM_FIXED 0x01
#define PNL_FM_FIXED_SIZE 0x02

/* cycle modes */
#define PNL_CM_NORMAL 0x00
#define PNL_CM_UNDER_CONSTRUCTION 0x01

/* drawing styles */
#define PNL_OPEN 0
#define PNL_FILLED 1

/* label placement (anti-clockwise from right) */
#define PNL_LABEL_RIGHT 0
#define PNL_LABEL_RIGHT_TOP 1
#define PNL_LABEL_UPPER_RIGHT 2
#define PNL_LABEL_TOP_RIGHT 3
#define PNL_LABEL_TOP 4
#define PNL_LABEL_TOP_LEFT 5
#define PNL_LABEL_UPPER_LEFT 6
#define PNL_LABEL_LEFT_TOP 7
#define PNL_LABEL_LEFT 8
#define PNL_LABEL_LEFT_BOTTOM 9
#define PNL_LABEL_LOWER_LEFT 10

```

panel.h

```

#define PNL_LABEL_BOTTOM_LEFT 11
#define PNL_LABEL_BOTTOM 12
#define PNL_LABEL_BOTTOM_RIGHT 13
#define PNL_LABEL_LOWER_RIGHT 14
#define PNL_LABEL_RIGHT_BOTTOM 15
#define PNL_LABEL_RIGHT_CENTER 16
#define PNL_LABEL_NORMAL 17

/* no offset, actually quite abnormal */

#define PNL_MAX_LABEL_TYPE 17

/* colors */

#define PNL_RGB_WHITE_COLOR 255, 255, 255 /* RGB triples */
#define PNL_RGB_BEVEL_LIGHT_COLOR 209, 236, 255
#define PNL_RGB_NORMAL_COLOR 125, 165, 223
#define PNL_RGB_OTHER_COLOR 62, 116, 207
#define PNL_RGB_BACKGROUND_COLOR 88, 134, 213
#define PNL_RGB_HIGHLIGHT_COLOR 0, 65, 191
#define PNL_RGB_BEVEL_DARK_COLOR 19, 42, 80
#define PNL_RGB_LABEL_COLOR 0, 0, 60
#define PNL_RGB_BLACK_COLOR 0, 0, 0

#define PNL_12BIT_WHITE_COLOR 30
#define PNL_12BIT_BEVEL_LIGHT_COLOR 229
#define PNL_12BIT_NORMAL_COLOR 178
#define PNL_12BIT_BACKGROUND_COLOR 173
#define PNL_12BIT_OTHER_COLOR 147
#define PNL_12BIT_HIGHLIGHT_COLOR 96
#define PNL_12BIT_HIGHLIGHT_COLOR 66
#define PNL_12BIT_BEVEL_DARK_COLOR 36
#define PNL_12BIT_LABEL_COLOR 31
#define PNL_12BIT_BLACK_COLOR 31

#define PNL_8BIT_WHITE_COLOR 30
#define PNL_8BIT_BEVEL_LIGHT_COLOR 29
#define PNL_8BIT_NORMAL_COLOR 23
#define PNL_8BIT_BACKGROUND_COLOR 20
#define PNL_8BIT_OTHER_COLOR 18
#define PNL_8BIT_HIGHLIGHT_COLOR 13
#define PNL_8BIT_BEVEL_DARK_COLOR 11
#define PNL_8BIT_LABEL_COLOR 9
#define PNL_8BIT_BLACK_COLOR 8

#define PNL_4BIT_WHITE_COLOR WHITE
#define PNL_4BIT_BEVEL_LIGHT_COLOR 15
#define PNL_4BIT_NORMAL_COLOR 14
#define PNL_4BIT_BACKGROUND_COLOR 13
#define PNL_4BIT_OTHER_COLOR 12
#define PNL_4BIT_HIGHLIGHT_COLOR 11
#define PNL_4BIT_BEVEL_DARK_COLOR 10
#define PNL_4BIT_LABEL_COLOR 9
#define PNL_4BIT_BLACK_COLOR BLACK

NL_EXTERN Colorindex pnl_white_color
NL_EXTERN Colorindex pnl_bevel_light_color
NL_EXTERN Colorindex pnl_normal_color
NL_EXTERN Colorindex pnl_background_color
NL_EXTERN Colorindex pnl_other_color
NL_EXTERN Colorindex pnl_highlight_color
NL_EXTERN Colorindex pnl_bevel_dark_color
NL_EXTERN Colorindex pnl_label_color
NL_EXTERN Colorindex pnl_black_color

/* mouse action sources */

#define PNL_SRC_QUEUE 0
#define PNL_SRC_SCRIPT 1

/* script message types */

#define PNL_MT_STATE 0xabc0
#define PNL_MT_MOUSE 0xabc1
#define PNL_MT_DELAY 0xabc2
#define PNL_MT_ENDBLOCK 0xabc3

/* function modes, ie, the context an actuator's function is called in */

#define PNL_FCNM_NONE 0
#define PNL_FCNM_DOWN 1
#define PNL_FCNM_ACTIVE 2
#define PNL_FCNM_UP 3

/* dopanel return modes, global control over dopanel return values */

#define PNL_DRM_RETURN_NULL 0
#define PNL_DRM_RETURN_PNL_CA 1

/* math relations */

#define PNL_HITRECT(x,y,x1,y1,x2,y2) \
    ((x)>(x1) && (x1)<(x2) && (y)>(y1) && (y)<(y2))

#define PNL_HITACT(a,wx,wy) \
    (a->beveled? \
        PNL_HITRECT((wx),(wy), \
            (a)->x-PNL_BEVEL_WIDTH, \
            (a)->y-PNL_BEVEL_WIDTH, \
            (a)->x+(a)->w+PNL_BEVEL_WIDTH, \
            (a)->y+(a)->h+PNL_BEVEL_WIDTH) : \
        PNL_HITRECT((wx),(wy), (a)->x, (a)->y, (a)->x+(a)->w, (a)->y+(a)->h))

#define ABS(a) ((a)>0.0?(a):- (a))
#define MIN(a, b) ((a)<(b)? (a):(b))
#define MAX(a, b) ((a)>(b)? (a):(b))
#define RANGE(a, b1, b2) \
    ((b1)<(b2)? \
        ((a)<(b1)? \
            (b1): \
            ((a)>(b2)? \
                (b2): \
                (a))) : \
        ((a)>(b2)? \
            (b2): \
            ((a)<(b1)? \
                (b1): \
                (a))))

#define PNL_VAL_TO_WORLD(val, min, max, dist) \
    (((val)-(min))/((max)-(min))+(min))* (dist)

#define PNL_WORLD_TO_VAL(x, min, max, dist) \
    ((x)/(dist))* ((max)-(min))+(min)

/* structure references */

#define PNL_ACCESS(type, ptr, member) (((type *) (ptr->data))->member)
#define PNL_MKUSERACT(struct_type, type) \
    (struct struct_type *) \
    _mkuseract (sizeof (struct struct_type), type)

#endif PNL_EDITOR_PARSING

```

panel.h

```

#define PNL_LABEL_BOTTOM_LEFT 11
#define PNL_LABEL_BOTTOM 12
#define PNL_LABEL_BOTTOM_RIGHT 13
#define PNL_LABEL_LOWER_RIGHT 14
#define PNL_LABEL_RIGHT_BOTTOM 15
#define PNL_LABEL_CENTER 16
#define PNL_LABEL_NORMAL 17
#define PNL_MAX_LABEL_TYPE 17

/* colors */
#define PNL_RGB_WHITE_COLOR 255, 255, 255 /* RGB triples */
#define PNL_RGB_BEVEL_LIGHT_COLOR 209, 236, 255
#define PNL_RGB_NORMAL_COLOR 125, 165, 223
#define PNL_RGB_OTHER_COLOR 62, 116, 207
#define PNL_RGB_BACKGROUND_COLOR 88, 134, 213
#define PNL_RGB_HIGHLIGHT_COLOR 0, 65, 191
#define PNL_RGB_BEVEL_DARK_COLOR 19, 42, 80
#define PNL_RGB_LABEL_COLOR 0, 0, 60
#define PNL_RGB_BLACK_COLOR 0, 0, 0

/* map indices */
#define PNL_12BIT_WHITE_COLOR 30
#define PNL_12BIT_BEVEL_LIGHT_COLOR 229
#define PNL_12BIT_NORMAL_COLOR 178
#define PNL_12BIT_BACKGROUND_COLOR 173
#define PNL_12BIT_OTHER_COLOR 147
#define PNL_12BIT_HIGHLIGHT_COLOR 96
#define PNL_12BIT_BEVEL_DARK_COLOR 66
#define PNL_12BIT_LABEL_COLOR 16
#define PNL_12BIT_BLACK_COLOR 31

#define PNL_8BIT_WHITE_COLOR 30
#define PNL_8BIT_BEVEL_LIGHT_COLOR 29
#define PNL_8BIT_NORMAL_COLOR 23
#define PNL_8BIT_BACKGROUND_COLOR 20
#define PNL_8BIT_OTHER_COLOR 18
#define PNL_8BIT_HIGHLIGHT_COLOR 13
#define PNL_8BIT_BEVEL_DARK_COLOR 11
#define PNL_8BIT_LABEL_COLOR 9
#define PNL_8BIT_BLACK_COLOR 8

#define PNL_4BIT_WHITE_COLOR 15
#define PNL_4BIT_BEVEL_LIGHT_COLOR 15
#define PNL_4BIT_NORMAL_COLOR 14
#define PNL_4BIT_BACKGROUND_COLOR 13
#define PNL_4BIT_OTHER_COLOR 12
#define PNL_4BIT_HIGHLIGHT_COLOR 11
#define PNL_4BIT_BEVEL_DARK_COLOR 10
#define PNL_4BIT_LABEL_COLOR 9
#define PNL_4BIT_BLACK_COLOR 8

/* structure references */
#define PNL_EXTERN Colorindex pnl_white_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_bevel_light_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_normal_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_background_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_other_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_highlight_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_bevel_dark_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_label_color PNL_INIT(0);
#define PNL_EXTERN Colorindex pnl_black_color PNL_INIT(0);

/* mouse action sources */
#define PNL_SRC_QUEUE 0
#define PNL_SRC_SCRIPT 1
/* script message types */
#define PNL_MT_STATE Oxabc0
#define PNL_MT_MOUSE Oxabc1
#define PNL_MT_DELAY Oxabc2
#define PNL_MT_ENDBLOCK Oxabc3

/* function modes, ie, the context an actuator's function is called in */
#define PNL_FCNM_NONE 0
#define PNL_FCNM_DOWN 1
#define PNL_FCNM_ACTIVE 2
#define PNL_FCNM_UP 3

/* dopanel return modes, global control over dopanel return values */
#define PNL_DRM_RETURN_NULL 0
#define PNL_DRM_RETURN_PNL_CA 1

/* math relations */
#define PNL_HITRECT(x,y,x1,y1,x2,y2) \
  ((x)>(x1) && (x)<(x2) && (y)>(y1) && (y)<(y2))
#define PNL_HITACT(a,wx,wy) \
  {a->beveled? \
    PNL_HITRECT((wx), (wy), \
      (a)->x-PNL_BEVEL_WIDTH, \
      (a)->y-PNL_BEVEL_WIDTH, \
      (a)->x+(a)->w+PNL_BEVEL_WIDTH, \
      (a)->y+(a)->h+PNL_BEVEL_WIDTH); \
    PNL_HITRECT((wx), (wy), (a)->x, (a)->y, (a)->x+(a)->w, (a)->y+(a)->h)}

#define ABS(a) ((a)>0.0?(a):- (a))
#define MIN(a, b) ((a)<(b)? (a): (b))
#define MAX(a, b) ((a)>(b)? (a): (b))
#define RANGE(a, b1, b2) \
  ((b1)<(b2)? \
    ((a)<(b1)? \
      (b1): \
      ((a)>(b2)? \
        (b2): (a))) \
    : ((a)<(b2)? \
      (b2): \
      ((a)>(b1)? \
        (b1): (a))))

#define PNL_VAL_TO_WORLD(val, min, max, dist) \
  (((val)-(min))/((max)-(min))*(dist))

#define PNL_WORLD_TO_VAL(x, min, max, dist) \
  ((x)/(dist))*((max)-(min))+(min)

/* structure references */
#define PNL_ACCESS(type, ptr, member) (((type *) (ptr->data))->member)
#define PNL_MKUSERACT(struct_type, type) \
  ((struct struct_type *) \
    _mkuseract (sizeof (struct struct_type), type))

#define PNL_EDITOR_PARSING

```

```

/* structure declarations */

typedef struct panel {
    short id; /* unique id */
    struct actuator *a; /* current actuator */
    struct actuator *al; /* actuator list */
    struct alist *autoalist; /* list of auto actuators */
    struct actuator *lastgroup; /* last actuator added to a group */
    Boolean active, selectable;
    long x, y, w, h; /* screen location of the window and its size */
    Coord maxx; /* bounding box enclosing all actuators and labels */
    Coord miny;
    Coord maxy;
    Coord cw, ch; /* char width and height */
    short gid; /* mex window number of this panel's window */
    short userid; /* mex window number of one of the user's windows */
    Object vobj; /* viewing transformations */
    float ppm; /* pixels per unit */
    char *label;
};

#define IRIS_4D
void (*delfunc)(struct panel *);
void (*fixfunc)(struct panel *);
else IRIS_4D
void (*delfunc)();
void (*fixfunc)();
void (*drawfunc)();
void (*downfunc)();
void (*upfunc)();
void (*dirtyfunc)();
char *u; /* pointer to arbitrary user data */
char *data; /* pointer to data peculiar to a particular actuator */
int datasize; /* size of data struct plus everything it points to */
Boolean automatic; /* true ==> newvalfunc called every dopanel */
Boolean selectable; /* false ==> unpickable, newvalfunc never called */
Boolean visible; /* does this actuator have a visible manifestation? */
Boolean beveled; /* is this actuator got a beveled edge? */
struct actuator *group;
struct actuator *next;
} Actuator;

#ifndef PNL_EDITOR_PARSING
typedef struct alist {
    Actuator *a;
    struct alist *next;
} Alist;

typedef struct {
    float x, y;
    Point;
} /* actuator specific data */

typedef struct {
    int firstpt, lastpt;
    Boolean Bind_Low, Bind_High;
    float *y;
    Actuator *lowlabel, *highlabel;
    Stripechart;
}

typedef struct {
    int mode;
    float linefactor;
    float differentialfactor;
    float valsave;
    Coord wsave;
    Coord bh;
    Slider;
} /* slider bar height */

typedef struct {
    int mode;
    int n;
    float linefactor;
    Coord wsave;
    Actuator *sa;
    Coord bh;
    Coord clrx, clry, clrw, clrh; /* expanded label clearing area */
} /* number of sliderbars */

```



```

void (*acttype)(); /* the init function for the slider bar act */
MultiSlider;

typedef struct {
    char *str;
    int len;
    TypeIn;
} Panel;

typedef struct {
    int mode;
    char *buf;
    char *delstr;
    int start;
    int dot;
    int mark;
    int col, lin;
    int len;
    int size;
    Coord cx, cw, cd;
    TypeOut;
} PanelData;

/* text to be displayed */
/* auto 'word' selection delimiters */
/* first char to display (appears in upper-left) */
/* insertion point */
/* other end of selection region */
/* width and height in character positions */
/* number of chars in buffer */
/* buffer size */
/* character dimensions */

PanelData *pnl; /* an incrementing index into the table */

PNL_EXTERN char *pnl_table[PNL_TAB:h_SIZE];
/* table of pointers to all panels and actuators */

PNL_EXTERN Panel *pnl_pl /* list of all control panels */
PNL_EXTERN AList *pnl_kl /* a list of actuators with key equivalents */
PNL_EXTERN Panel *pnl_cp /* panel being moused (current panel) */
PNL_EXTERN Actuator *pnl_ca /* actuator being moused (current actuator) */
PNL_EXTERN Panel *pnl_cp_save /* last cp from queue */
PNL_EXTERN Actuator *pnl_ca_save /* last ca from queue */
PNL_EXTERN Boolean pnl_ca_active_save /* state of active field in pnl_ca_save */
PNL_EXTERN Boolean pnl_cp_active_save /* state of active field in pnl_cp_save */
PNL_EXTERN Actuator *pnl_mouse_act /* the mouse actuator, if any */
PNL_EXTERN ScreenCoord pnl_mx, pnl_my;
PNL_EXTERN Coord pnl_x, pnl_y; /* screen coords if outside a panel */
PNL_EXTERN long int pnl_frame_number /* frames since last delay message */
PNL_EXTERN int pnl_delay /* frames 'z1', next message allowed from the queue */
PNL_EXTERN Boolean pnl_delay_virgin /* flag prevents writing initial delay token */
PNL_EXTERN Boolean pnl_readscript /* pnl_init(FALSE); */

```

```

/* true when reading a script */
PWL_EXTERN Boolean pnl_writescript PNL_INIT(FALSE);
/* true when writing a script */
PWL_EXTERN int pnl_scriptinfd PNL_INIT(0);
PWL_EXTERN int pnl_scriptoutfd PNL_INIT(0);
PWL_EXTERN char *pnl_scriptinfile PNL_INIT("panel.script");
PWL_EXTERN char *pnl_scriptoutfile PNL_INIT("panel.s.script");
PWL_EXTERN Boolean pnl_virgin PNL_INIT(TRUE);
/* haven't done a dpanel yet */
PWL_EXTERN Boolean pnl_saveuserredraw PNL_INIT(FALSE);
/* if the user is using them, requeue them */
PWL_EXTERN ScreenCoord pnl_ox, pnl_oy;
/* last window origin */
PWL_EXTERN Boolean pnl_justdown;
/* true when button first goes down */
PWL_EXTERN Boolean pnl_justup;
/* true when button first goes up */
PWL_EXTERN Boolean pnl_mousedown;
/* true when button is (was) down */
/* (according to the queue) */
PWL_EXTERN Boolean pnl_shiftck;
PWL_EXTERN Boolean pnl_init(FALSE);
/* true when the shift key is down */
PWL_EXTERN Boolean pnl_controlkey PNL_INIT(FALSE);
/* true when the control key is down */
PWL_EXTERN int pnl_winsave;
/* old of (user's) window when dpanel() is called */
PWL_EXTERN int pnl_action_source PNL_INIT(PNL_SRC_QUEUE);
/* where last mouse action came from */
PWL_EXTERN float pnl_char_threshold PNL_INIT(PNL_CHAR_THRESHOLD);
/* see PNL_CHAR_THRESHOLD above */
PWL_EXTERN Boolean pnl_dontdraw PNL_INIT(FALSE);
/* don't update panels (to avoid swapbuffers) */
PWL_EXTERN Boolean pnl_beveled PNL_INIT(TRUE);
/* global control for disabling drawing bevels */
PWL_EXTERN int pnl_furcmode PNL_INIT(PNL_FCMN_NONE);
/* role of called -p-, active-, or downfuns */
PWL_EXTERN Boolean pnl_ignore_delay PNL_INIT(FALSE);
/* ignore delay packets when reading script */
PWL_EXTERN Boolean pnl_panelbell PNL_INIT(TRUE);
/* ring bell for hits on unselectable panels */
PWL_EXTERN int pnl_dopanel_return_mode PNL_INIT(PNL_DRM_RETURN_PNL_CA);
/* to ground out dpanel() when it's build mode */

```

```

#define PNL_FADE_PATTERN_INDEX 1
#define PNL_FADE_PATTERN_SIZE 16
#define PNL_FADE_PATTERN
/* wide diagonals */
{ 0x0000, 0x7878, 0x3c3c, 0x1e1e,
  0x0f0f, 0x7878, 0x3c3c, 0x1e1e,
  0x0f0f, 0x7878, 0x3c3c, 0x1e1e }

/* line diagonals */
{ 0x9999, 0xcccc, 0x6666, 0x3333,
  0x9999, 0xcccc, 0x6666, 0x3333,
  0x9999, 0xcccc, 0x6666, 0x3333 }

/* pattern used to cover unselectable actuators */
{ 0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f,
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f,
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f }

{ 0x5555, 0x5555, 0x5555, 0x5555,
  0x5555, 0x5555, 0x5555, 0x5555,
  0x5555, 0x5555, 0x5555, 0x5555 }

/* function declarations */
/* panel library utilities */

#ifdef IRIS_4D
void *alloc(int);
void *pnl_alloc(int);
#else IRIS_4D
void *alloc();
void *pnl_alloc();
#endif IRIS_4D
/* user functions */

PNL_EXTERN Panel *mkpanel();
PNL_EXTERN Actuator *dopanel();
PNL_EXTERN void dumpstate();
PNL_EXTERN void drawpanel();
PNL_EXTERN short userredraw();
PNL_EXTERN void needredraw();
PNL_EXTERN Boolean dumppanel();

PNL_EXTERN Panel *pnl_mkpanel();
PNL_EXTERN Actuator *pnl_dopanel();
PNL_EXTERN void pnl_dumpstate();
PNL_EXTERN void pnl_drawpanel();
PNL_EXTERN short pnl_userredraw();
PNL_EXTERN void pnl_needredraw();
PNL_EXTERN Boolean pnl_dumppanel();

#ifdef IRIS_4D
PNL_EXTERN Actuator *mkact(/*void (*)()*/); /* parameter 'decl' doesn't seem to work */
PNL_EXTERN Actuator *mkuseract(/*int, void (*)()*/); /* parameter 'decl' doesn't seem to w

```

panel.h

```

/*
 *L_EXTERN void newvalact(/**tuator *,Panel *,Coord, Coord*/);
 *L_EXTERN void adact(Actuator *,Panel *);
 *L_EXTERN void adsubact(Actuator *,Actuator *);
 *L_EXTERN void delact(Actuator *);
 *L_EXTERN void endgroup(Panel *);
 *L_EXTERN void addtogroup(Actuator *,Panel *);
 *L_EXTERN void fixpanel(Panel *);
 *L_EXTERN void fixact(Actuator *);
 *L_EXTERN void labeloffsets(Actuator *);
 *L_EXTERN void labeldimensions(Actuator *);
 *L_EXTERN char *g_gets(ColorIndex,ColorIndex,ColorIndex,ColorIndex,char *,int);
 *L_EXTERN Boolean pnl_beginreadscript(char *);
 *L_EXTERN Boolean pnl_beginwritscript(char *);
 *L_EXTERN Boolean pnl_beginappendscript(char *);
 *L_EXTERN void pnl_endreadscript();
 *L_EXTERN void pnl_endwritscript();
 *L_EXTERN float pnl_stwidth(Panel *, char **);
 *L_EXTERN void pnl_listadd(AList **, Alist **);
 *L_EXTERN void pnl_listdelete(AList **, Alist **);
 *L_EXTERN Boolean pnl_listin(Alist **, Alist *);

 *L_EXTERN Actuator *pnl_mkact(/**void (*)()*/); /* parameter decl doesn't seem
 to work */
 *L_EXTERN Actuator *pnl__mkusact(/**int, void (*)()*/); /* parameter decl doesn't seem
 to work */
 *L_EXTERN void pnl_newvalact(/**Actuator *,Panel *,Coord, Coord*/);
 *L_EXTERN void pnl_adact(Actuator *,Panel *);
 *L_EXTERN void pnl_adsubact(Actuator *,Actuator *);
 *L_EXTERN void pnl_delact(Actuator *);
 *L_EXTERN void pnl_endgroup(Panel *);
 *L_EXTERN void pnl_addtogroup(Actuator *,Panel *);
 *L_EXTERN void pnl_fixpanel(Panel *);
 *L_EXTERN void pnl_fixact(Actuator *);
 *L_EXTERN void pnl_labeloffsets(Actuator *);
 *L_EXTERN void pnl_labeldimensions(Actuator *);
 *L_EXTERN char *g_gets(ColorIndex,ColorIndex,ColorIndex,ColorIndex,char *,int);
 *L_EXTERN Boolean pnl_beginreadscript(char *);
 *L_EXTERN Boolean pnl_beginwritscript(char *);
 *L_EXTERN Boolean pnl_beginappendscript(char *);
 *L_EXTERN void pnl_endreadscript();
 *L_EXTERN void pnl_endwritscript();
 *L_EXTERN float pnl_stwidth(Panel *, char **);
 *L_EXTERN void pnl_listadd(AList **, Alist **);
 *L_EXTERN void pnl_listdelete(AList **, Alist **);
 *L_EXTERN Boolean pnl_listin(Alist **, Alist *);

 *L_EXTERN void pnl_slider(Actuator *);
 *L_EXTERN void pnl_vslider(Actuator *);
 *L_EXTERN void pnl_hslider(Actuator *);
 *L_EXTERN void pnl_dvslider(Actuator *);
 *L_EXTERN void pnl_dhslider(Actuator *);
 *L_EXTERN void pnl_filled_slider(Actuator *);
 *L_EXTERN void pnl_filled_vslider(Actuator *);
 *L_EXTERN void pnl_filled_hslider(Actuator *);
 *L_EXTERN void pnl_button(Actuator *);
 *L_EXTERN void pnl_wide_button(Actuator *);
 *L_EXTERN void pnl_toggle_button(Actuator *);
 *L_EXTERN void pnl_radio_button(Actuator *);
 *L_EXTERN void pnl_left_arrow_button(Actuator *);
 *L_EXTERN void pnl_right_arrow_button(Actuator *);
 *L_EXTERN void pnl_up_arrow_button(Actuator *);
 *L_EXTERN void pnl_down_arrow_button(Actuator *);
 *L_EXTERN void pnl_left_double_arrow_button(Actuator *);
 *L_EXTERN void pnl_right_double_arrow_button(Actuator *);
 *L_EXTERN void pnl_up_double_arrow_button(Actuator *);
 *L_EXTERN void pnl_down_double_arrow_button(Actuator *);
 *L_EXTERN void pnl_meter(Actuator *);
 *L_EXTERN void pnl_analog_meter(Actuator *);
 *L_EXTERN void pnl_strip_chart(Actuator *);

#endif IRIS_4D

/* initialization functions */

#define IRIS_4D
extern void pnl_slider(Actuator *);
extern void pnl_vslider(Actuator *);
extern void pnl_hslider(Actuator *);
extern void pnl_dvslider(Actuator *);
extern void pnl_dhslider(Actuator *);
extern void pnl_filled_slider(Actuator *);
extern void pnl_filled_vslider(Actuator *);
extern void pnl_filled_hslider(Actuator *);
extern void pnl_button(Actuator *);
extern void pnl_wide_button(Actuator *);
extern void pnl_toggle_button(Actuator *);
extern void pnl_radio_button(Actuator *);
extern void pnl_left_arrow_button(Actuator *);
extern void pnl_right_arrow_button(Actuator *);
extern void pnl_up_arrow_button(Actuator *);
extern void pnl_down_arrow_button(Actuator *);
extern void pnl_left_double_arrow_button(Actuator *);
extern void pnl_right_double_arrow_button(Actuator *);
extern void pnl_up_double_arrow_button(Actuator *);
extern void pnl_down_double_arrow_button(Actuator *);
extern void pnl_meter(Actuator *);
extern void pnl_analog_meter(Actuator *);
extern void pnl_strip_chart(Actuator *);

```

```

>>>tern void pnl_scale_chart(Actorator *);
>>>tern void pnl_puck(Actorator *);
>>>tern void pnl_floating_puck(Actorator *);
>>>tern void pnl_rubber_puck(Actorator *);
>>>tern void pnl_typein(Actorator *);
>>>tern void pnl_typeout(Actorator *);
>>>tern void pnl_label(Actorator *);
>>>tern void pnl_mouse(Actorator *);
>>>tern void pnl_sliderold(Actorator *);
>>>tern void pnl_palette(Actorator *);
>>>tern void pnl_vpalette(Actorator *);
>>>tern void pnl_hpalette(Actorator *);
>>>tern void pnl_dial(Actorator *);
>>>tern void pnl_multislider(Actorator *);
>>>tern void pnl_hmultislider(Actorator *);
>>>tern void pnl_vmultislider(Actorator *);
>>>tern void pnl_multislider_bar(Actorator *);
>>>tern void pnl_multislider_open_bar(Actorator *);
>>>tern void pnl_vmultislider_bar(Actorator *);
>>>tern void pnl_hmultislider_bar(Actorator *);
>>>tern void pnl_hmultislider_open_bar(Actorator *);
>>>tern void pnl_menu(Actorator *);
>>>tern void pnl_menu_menu(Actorator *);
>>>tern void pnl_sub_menu(Actorator *);
>>>tern void pnl_menu_item(Actorator *);
>>>tern void pnl_icon(Actorator *);
>>>tern void pnl_cycle(Actorator *);
>>>tern void pnl_scroll(Actorator *);
else IRIS_4D
>>>tern void pnl_slider();
>>>tern void pnl_vslider();
>>>tern void pnl_hslider();
>>>tern void pnl_dvslider();
>>>tern void pnl_dhslider();
>>>tern void pnl_filled_slider();
>>>tern void pnl_filled_vslider();
>>>tern void pnl_filled_hslider();
>>>tern void pnl_button();
>>>tern void pnl_wide_button();
>>>tern void pnl_toggle_button();
>>>tern void pnl_radio_button();
>>>tern void pnl_left_arrow_button();
>>>tern void pnl_right_arrow_button();
>>>tern void pnl_up_arrow_button();
>>>tern void pnl_down_arrow_button();
>>>tern void pnl_double_arrow_button();
>>>tern void pnl_double_arrow_button();
>>>tern void pnl_meter();
>>>tern void pnl_analog_meter();
>>>tern void pnl_analog_bar();
>>>tern void pnl_strip_chart();
>>>tern void pnl_scale_chart();
>>>tern void pnl_puck();
>>>tern void pnl_floating_puck();
>>>tern void pnl_rubber_puck();
>>>tern void pnl_typein();
>>>tern void pnl_typeout();
>>>tern void pnl_label();
>>>tern void pnl_sliderold();

```